Attorney Docket No.: 018926-011310US
Client Reference No. D3074

PATENT APPLICATION

SINGLE INSTRUCTION MULTIPLE DATA IMPLEMENTATION OF FINITE IMPULSE RESPONSE FILTERS INCLUDING ADJUSTMENT OF RESULT

INVENTOR(S):

Scott Contini,
a citizen of the USA residing at
3956 Texas St, Apt 13
San Diego, CA 92104

Chanchal Chatterjee,
a citizen of the USA residing at
10553 Harvest View Way
San Diego, CA 92128

ASSIGNEE:

General Instrument Corporation
101 Tournament Drive
Horsham, PA 19044

ENTITY:

Large

PATENT APPLICATION

SINGLE INSTRUCTION MULTIPLE DATA IMPLEMENTATION OF FINITE IMPULSE RESPONSE FILTERS INCLUDING ADJUSTMENT OF RESULT

COPYRIGHT NOTICE

Cross- References To Related Applications

[02] This application is related to the following co-pending U.S. Patent Applications which are hereby incorporated by reference as if set forth in full in this specification:

Serial No. [TBD], filed on [TBD], entitled "SINGLE INSTRUCTION MULTIPLE DATA IMPLEMENTATIONS OF FINITE IMPULSE RESPONSE FILTERS"; and

Serial No. 10/057,694, filed on January 23, 2002, entitled "METHODS FOR EFFICIENT FILTERING OF DIGITAL SIGNALS."

Background Of The Inv ntion

Field of the Invention

[03] This invention is related in general to computer processing and more specifically to the use of single instruction multiple data (SIMD) instructions to achieve finite impulse response filter operations in a digital processor.

Description Of The Background Art

[04] Finite Impulse Response (FIR) filter operations are an important type of digital computation or processing. FIR filters are commonly used, for example, in pre-processing, post-processing, motion compensation, and motion estimation for video compression standards. The implementation of FIR filters in computer programs, or other digital processing approaches, is useful in many other applications including audio processing, signal conditioning, simulation of electronic components, etc.

[05] FIR filter operations can be very demanding on digital processing systems because of the large number of iterative operations that must be performed very quickly. The number of operations, speed of operation, resolution of coefficient values, and other factors all contribute to the accuracy of the implementation and the amount of processing resources that are necessary to achieve a design goal. In this respect, a slight advantage in FIR filter operations that are executed frequently (i.e., in an "inner loop" of a program) can result in very significant performance gains.

[06] An FIR filter that is of special interest in video compression and encoding techniques is referred to as a *transversal* or *tapped delay* filter. These filters multiply a set of coefficients to pixel values of a video frame to generate a new pixel value. Such an operation is useful, for example, to compress an image by combining adjacent pixel values into a smaller number of pixel values. Typically, this type of FIR filter includes only positive coefficients.

[07] Fig. 1 illustrates four pixel values a_1, a_2, a_3, and a_4. Subpixel b is desired to be the average of the four pixels computed as:

[08] $$b = (a_1 + a_2 + a_3 + a_4 + 2) >> 2, \qquad (1)$$

where >> is a bitwise right shift operator.

[09] In a typical application where pixel values are limited to values in the range 0-255, the pixels $a_1,\ldots,a_4$ are each represented in one byte or 8 bits. Thus, a total of four bytes is necessary to operate on the four pixel values at once.

[10] Typically, a single frame in a digital video presentation of moderate resolution can include 600x800 = 480,000 pixels. Such a frame might be displayed 30 times per second. Moreover, it may be necessary to perform additional "passes" over the frame so that, for example, in subsequent passes the pixels, themselves, are combined into subpixels to further compress an image. Thus, numerous subpixel computations may be necessary. Further digital video formats, such as high-definition television, use much higher screen resolutions and color depths. It should be apparent that such filter operations could place enormous requirements on processing resources, especially when the operations must be performed in real time.

[11] One approach that the prior art uses to provide increased efficiency in filter or array operations is to use Single Instruction Multiple Data (SIMD) instructions. Such instructions allow value-packing, byte-packing, or other concatenating of values into a single word or other unit of data. The unit of data can be processed quickly by performing a desired operation in parallel on the packed values.

[12] SIMD-type instructions are available in many processors. Examples include Intel Multi-Media Extensions (MMX)^{TM} and Streaming SIMD Extension (SSE)^{TM}, as well as NEC VR5432, Equator MAP-CA^{TM}, and Philips TM-1300 processors. In processors whose architecture supports SIMD instructions there are typically multiple identical processors, N, each with its own local memory where it can store data. All processors work under the control of a single instruction stream issued by a central control unit. There are typically N data streams, one per processor. The processors operate synchronously: at each step, all processors execute the same instruction on a different data element. This architecture allows N computations in parallel. Thus, if N=8, it is possible to achieve a computational speedup of 8.

[13] Fig. 2 provides an example of the operation of a SIMD instruction. The SIMD instruction performs an operation, "OP," on two sets of data: $A=[a_1,\ldots,a_8]$, a vector of 8 data values, each of which is an *unsigned 8-bit integer*, i.e., $a_i \in [0,255]$; and $B=[b_1,\ldots,b_8]$, another vector of unsigned integers within the range [0,255]. The final result $C=[c_1,\ldots,c_8]$ is achieved by simultaneously operating on all 8 values of a_i and b_i as $c_i = a_i$ OP b_i, for $i=1,\ldots,8$. In this example, A, B and C are 64-bit registers in which all 8 values of a_i, b_i, and c_i are packed as contiguous bytes as shown in Figure 2, i.e., $N=8$. Such operations are also known as *packed operations*, since 8 values of data are packed in a single register A, B or C.

[14] One specific type of operation of interest in filter operations is the *PAVG* operation that can be found, e.g., in the Intel MMX™ instruction set. The *PAVG* instruction performs the following computation:

[15] $$PAVG(A,B) = [(a_i+b_i+1) >> 1, i=1,\ldots,8]. \qquad (2)$$

[16] This operation takes 8-bit values of a_i, b_i, and stores the intermediate sum (a_i+b_i+1) in 9 bits before doing bitwise logical right shift operation to get the final result. It is available in many processors, including the ones mentioned above, and uses only one instruction. This instruction has the latency of 1 clock cycle in the Intel Pentium III, 2 clock cycles in Intel Pentium 4, and Advanced Micro Device's (AMD's) Athlon, with a throughput of 1 clock cycle. The same performance is realized for other operations in these architectures, such as packed addition (+), subtraction (–), bitwise *AND* (&), bitwise *OR* (|), bitwise *EXCLUSIVE-OR* (^), bitwise right shift (>>), and bitwise left shift (<<) operations.

[17] Although SIMD instructions can improve the efficiency and speed of computations, such instructions are sometimes difficult to use effectively when the SIMD instructions do not provide the exact type of operation needed. For example, as stated above, *PAVG* computes $(a_i+b_i+1) >> 1$. An average of two vectors <u>rounded up</u>. However, it is more desirable in some filter operations to obtain $(a_i+b_i)>>1$, which is a <u>truncated</u> average where the remainder, or fractional part, is discarded. Such a difference in operation is significant where multiple passes of frame data are made as the average intensity value of subpixels may increase and result in artifacts or other objectionable qualities to the processed data. In the architectures discussed herein, a SIMD instruction to compute $(a_i+b_i) >> 1$ is not provided. Typically, a non-SIMD approach must be used.

[18] A problem also arises when the number of arguments required by a SIMD operation is not the same as the number of variables in a formula to be implemented by the SIMD operation. For example, if a SIMD instruction accepts two arguments then it is "mismatched" to implement a formula, computation or operation with more than two variables or values. The same can be said, for example, for a SIMD instruction with three arguments used to implement a formula with other than three variables, etc.

[19] Fig. 3 illustrates a non-SIMD approach to compute $(a_i+b_i)>>1$.

[20] In Fig. 3, a_i, b_i are unsigned integers within the range [0,255], i.e., each a_i, b_i is represented in 8 bits. The number of processors, $N=8$, i.e., the operation $(a_i+b_i)>>1$ is simultaneously performed on 8 values of a_i and b_i for $i=1,\ldots,8$. All 8 values of a_i (usually contiguous pixels) are packed in 64-bit register, A, and 8 values of b_i in 64-bit register B. Since a_i+b_i can exceed 8 bits, the 8-bit (byte) values of a_i, b_i are unpacked into 16-bits (words) as four 16-bit values per 64-bit register,. Then the packed registers A and B are added together, followed by bitwise logical right shift by 1, followed by packing again. Note that in most processors, data can be packed into 64-bit registers as 8 (byte), 16 (word), 32 (dword), or 64 (qword) bit values only. Figure 3 shows the conventional method of doing the packed operation $c_i=(a_i+b_i)>>1$ for $i=1,\ldots,8$. It is clear from Figure 3, that given sufficient memory, 9 instructions are needed to achieve the result $c_i=(a_i+b_i)>>1$ for all 8 values of a_i and b_i. Each instruction in Figure 3 is represented by an ellipse.

Summary of Embodiments of the Invention

[21] The invention provides improved results in some cases of digital calculation of finite impulse response (FIR) filters. A preferred embodiment of the invention is applied to techniques for FIR calculation discussed in the co-pending patent application entitled "SINGLE INSTRUCTION MULTIPLE DATA IMPLEMENTATIONS OF FINITE IMPULSE RESPONSE FILTERS," referenced, above. In the co-pending patent application a system for efficient derivation of FIR values is presented using single-instruction multiple data (SIMD) types of operations. In a preferred embodiment, the results of the FIR calculations are subjected to additional operations using a SIMD instruction called *PAVG*. The results of *PAVG* are a rounded-up average of two sets of packed values. Adjustments are made on the rounded-up average to obtain an exact desired result for various filter calculations, or to obtain results within a desired error range, or results that do not exceed, or fall below, desired values in relation to the exact answer. Various techniques for minimizing processor resources (e.g., processing cycles, memory) are presented.

[22] These provisions together with the various ancillary provisions and features which will become apparent to those artisans possessing skill in the art as the following description proceeds are attained by devices, assemblies, systems and methods of embodiments of the present invention, various embodiments thereof being shown with reference to the accompanying drawings, by way of example only, wherein:

[23] One embodiment of the invention provides [@@]

Brief Description of the Drawings

[24] Fig. 1 illustrates a subpixel average of four pixel values;

[25] Fig. 2 shows an example of the execution of a single-instruction multiple-data (SIMD) instruction;

[26] Fig. 3 shows a non-SIMD approach to a calculation; and

[27] Fig. 4 shows a SIMD implementation with adjustment.

Detailed Description of Embodiments of the Invention

[28] The technique of the present invention includes adjusting an FIR calculation result using SIMD instructions to obtain an improved result. This technique is the focus of section 5 of this specification. Other sections include text from the co-pending patent application entitled "SINGLE INSTRUCTION MULTIPLE DATA IMPLEMENTATIONS OF FINITE IMPULSE RESPONSE FILTERS," cited, above, upon whose results the adjustments of the present invention are based.

[29] A preferred embodiment of the invention uses Intel's MMX/SSE architecture, including the SIMD *PAVG* operation. Other embodiments may use other processors, instructions and operations in a manner similar to that disclosed herein and realize similar computational benefits. In addition, other techniques and approaches for performing processing may benefit from one or more of the features presented herein, such as the techniques of the related patent application "METHODS FOR EFFICIENT FILTERING OF DIGITAL SIGNALS," cited above.

[30] Table I shows notations used in this application.

Operator	**Description**
+	Addition
–	subtraction
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR

>>	bitwise logical right shift
<<	bitwise logical left shift
~	Bitwise NOT
CLIP(x)	Clips x to range [0,255]
ODD(x)	Returns 1 when x is odd, 0 otherwise
EVEN(x)	Returns 1 when x is even, 0 otherwise

TABLE I

[31] The present invention allows computing $c_i=(a_i+b_i)>>1$ for $i=1,\ldots,8$, in an efficient manner using a SIMD instruction such as *PAVG*. Note that simply using the *PAVG* instruction on packed values in registers A and B will not yield the correct answer. For example, when a_i+b_i is an odd number $PAVG(a_i, b_i)$ gives a result that is one more than the correct answer. The result of a *PAVG* operation must be adjusted as follows:

$$C = PAVG(\text{A, B}) - (\text{A} \wedge \text{B})\ \&\ 0\text{x}01,$$

where 0x01 is a 8-bit number whose least significant bit is 1 and the rest are 0's.

[32] The *PAVG* operation with adjustment is shown in Fig. 4. Assuming sufficient memory, only 4 instructions instead of the previous 9 instructions (without using *PAVG*) are needed to achieve the packed operation $C=(A+B)>>1$. This is an approximate speedup of 9/4 = 2.25 times.

[33] A preferred embodiment of the invention achieves the same computational result as in Fig. 4 with even fewer instructions by appropriately using the *PAVG* instruction in combination with supplemental logical operations to adjust for the rounded-up average. As described below, several FIR filtering operations can be modified to obtain result in fewer instructions when compared to conventional SIMD implementations.

[34] Without loss of generality, let $A_1, A_2,\ldots,A_{16}$ be 16 vectors, each of which contain 8 packed data elements. For example, A_5 contains 8 data elements $A_5 = [a_{(5,1)}, \ldots, a_{(5,8)}]$. Each data element $a_{(1,i)}, \ldots, a_{(16,i)}$ for $i=1,\ldots 8$, is within the range [0,255], i.e., they are represented by bytes, and $A_1,\ldots,A_{16}$ are packed 64-bit registers:

$$A_j = [a_{(j,1)}, \ldots, a_{(j,8)}] \text{ for } j=1,\ldots,16. \qquad (4)$$

[35] We perform various operations on the packed 64-bit registers $A_1,\ldots,A_{16}$ to obtain different FIR filters described below. We define packed 64-bit vectors/registers *ONE* and ONE_4 as follows:

$$ONE = [0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01],$$
$$ONE_4 = [0x0001, 0x0001, 0x0001, 0x0001], \quad (5)$$

where 0x01 is a byte containing 1 in its least significant bit and 0's elsewhere. The packed 64-bit register *ONE* contains 8 packed bytes, each containing 0x01. On the other hand, the packed 64-bit register ONE_4 contains 4 packed words (16 bits), each containing 0x0001.

[36] The FIR filters used in a preferred embodiment include:

1. **Type 1 Filter:** $(A_1+A_2+c*ONE) >> 1$, where $c \in \{-2,-1,0,1,2\}$, (6)
2. **Type 2 Filter:** $(A_1+A_2+A_3+A_4+c*ONE) >> 2$, where $c \in \{0,1,2\}$, (7)
3. **Type 3 Filter:** $(A_1+A_2+A_3+A_4+A_5+A_6+A_7+A_8+c*ONE) >> 3$, where $c \in \{0,1,2,3,4\}$, (8)
4. **Type 4 Filter:** $(A_1+A_2+\ldots+A_{15}+A_{16}+c*ONE) >> 4$, where $c \in \{0,1,2,3,4,5,6,7,8\}$. (9)

[37] All 4 types of FIR filters are useful for video compression applications. There are numerous FIR filters that can be constructed from these 4 basic types, in addition to those described herein. For example, the filter $(2A_1+A_2+A_3+2*ONE) >> 2$ is a Type 2 filter with $A_1=A_4$. Similarly, the filter $(A_1+2A_2+2A_3+2A_4+A_5+4*ONE) >> 3$ is a Type 3 filter with $A_2=A_6$, $A_3=A_7$, and $A_4=A_8$. Many other types of filters can be constructed as will be apparent to one of skill in the art.

[38] Instructions according to the present invention can be used to obtain exact filter computations. Such exactness may be necessary as, e.g., in motion compensation and estimation applications where accuracy is key. In other cases an approximation of the filter computation may be sufficient. For example, in cases where the number of operations is large an approximate computation can be a better tradeoff. The approximations of the preferred embodiments produce an error of ± 1 in the final result for a small percentage of all values of $a_{(j,i)} \in [0,255]$ for $i=1,\ldots,8$, and $j=1,\ldots,16$. These results are useful in cases such as post processing, where a small error of ± 1 (in intensity

or color value) is inconsequential in the final result. Naturally, other approximations of different degrees of accuracy are possible and are within the scope of the invention.

I. Type 1 FIR Filters

[39] There are 5 variations of the Type 1 FIR filters (A_1+A_2+c*ONE), where

$c \in \{-2, -1,0,1,2\}$, based on the 5 choices of constant c. We state the SIMD implementation for each of these filters:

$$(A_1+A_2-2*ONE) >> 1 = PAVG(A_1,A_2) - ONE - (A_1 \wedge A_2) \,\&\, ONE, \tag{10}$$

$$(A_1+A_2-ONE) >> 1 = CLIP(PAVG(A_1,A_2) - ONE), \tag{11}$$

$$(A_1+A_2) >> 1 = PAVG(A_1,A_2) - (A_1 \wedge A_2) \,\&\, ONE, \tag{12}$$

$$(A_1+A_2+ONE) >> 1 = PAVG(A_1,A_2), \tag{13}$$

$$(A_1+A_2+2*ONE) >> 1 = PAVG(A_1,A_2) + (\sim(A_1 \wedge A_2) \,\&\, ONE). \tag{14}$$

[40] There is a less efficient solution for $(A_1+A_2) >> 1$ that will be used to simplify expressions:

$$(A_1+A_2) >> 1 = (A_1 >> 1) + (A_2 >> 1) + (A_1 \,\&\, A_2 \,\&\, ONE). \tag{15}$$

[41] Although (15) uses more instructions that (12), we need this expression to evaluate other filters. In (15), $(A_1 \,\&\, A_2 \,\&\, ONE)$ is a correction term that is necessary when both A_1 and A_2 contain odd integers. An approximate solution for $(A_1+ A_2) >> 1$ is:

$$(A_1+A_2) >> 1 \cong PAVG(CLIP(A_1-ONE),A_2) \text{ or } PAVG(A_1,CLIP(A_2-ONE)). \tag{16}$$

[42] In most processors, subtract and CLIP() can be realized in one instruction. So the implementations in (16) require only 2 instructions.

II. Type 2 FIR Filters

[43] There are 3 variations of Type 2 filters (7) based on the 3 choices of constant c, where $c \in \{0,1,2\}$. We show the derivation of each filter. We define the following 64-bit packed registers, each containing 8 data elements of one byte each:

$$B_1 = PAVG(A_1,A_2), B_2 = PAVG(A_3,A_4), EB_1 = (A_1 \wedge A_2), EB_2 = (A_3 \wedge A_4). \tag{17}$$

A. Type 2, Filter 1: R = $(A_1 + A_2 + A_3 + A_4 + 2*ONE) >> 2$

i. Conventional SIMD Solution

[44] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (4 Instructions) A_{iL} = Unpack Low 4 Bytes of A_i, for $i = \{1,2,3,4\}$,
2. (4 Instructions) A_{iH} = Unpack High 4 Bytes of A_i, for $i = \{1,2,3,4\}$,

3. (5 Instructions) Add and Shift lower 4 words of $A_1,\ldots,A_4$ to obtain lower 4 words of R_L as:

   $R_L = (A_{1L} + A_{2L} + A_{3L} + A_{4L} + 2{*}ONE_4) >> 2,$
4. (5 Instructions) Add and Shift higher 4 words of $A_1,\ldots,A_4$ to obtain higher 4 words of R_H as:

   $R_H = (A_{1H} + A_{2H} + A_{3H} + A_{4H} + 2{*}ONE_4) >> 2,$
5. (1 Instruction) Pack R_H and R_L into final register R.

We require 19 instructions to perform this filter by conventional SIMD methods.

ii. Efficient SIMD Solution

[45] In order to implement this filter efficiently, we simplify as follows:

$$R = (((A_1+A_2+ONE)>>1) + ((A_3+A_4+ONE)>>1) + E) >> 1 = (B_1+B_2+E) >> 1, \qquad (18)$$

where E is the correction term that is necessary when both (A_1+A_2+ONE) and (A_3+A_4+ONE) are odd integers as in (15). Detection of odd or even integers is performed with the functions $ODD()$ and $EVEN()$. Where $ODD()$ returns "1" for each packed argument value only if the packed argument value is an odd number and returns "0" otherwise, and where $EVEN()$ returns "1" for each packed argument value only if the packed argument value is an even number and returns "0" otherwise.

$$\begin{aligned} E &= ODD(A_1+A_2+ONE)\ \&\ ODD(A_3+A_4+ONE) = EVEN(A_1+A_2)\ \&\ EVEN(A_3+A_4) \\ &= {\sim}(A_1 \wedge A_2)\ \&\ {\sim}(A_3 \wedge A_4)\ \&\ ONE = {\sim}(EB_1 \mid EB_2)\ \&\ ONE. \end{aligned} \qquad (19)$$

We note that $E \in \{0,1\}$. From (18) and (12), we have:

$$R = \begin{cases} PAVG(B_1,B_2) - (B_1 \wedge B_2)\,\&\,ONE & \text{when } E = 0 \\ PAVG(B_1,B_2) & \text{when } E = 1 \end{cases}. \qquad (20)$$

We simplify (20) as:

$$R = PAVG(B_1,B_2) - (B_1 \wedge B_2)\ \&\ {\sim}E\ \&\ ONE,$$

which is same as:

$$R = PAVG(B_1,B_2) - (B_1 \wedge B_2)\ \&\ ((A_1 \wedge A_2) \mid (A_3 \wedge A_4))\ \&\ ONE. \qquad (21)$$

The solution in (21) requires 10 instructions. We have an approximate 19:10 (approx. 2:1) speedup by using (21).

iii. Approximate SIMD Solution

[46] Besides the accurate solution, we can obtain an approximate solution in fewer instructions by assuming the least significant bit of EB_1 or EB_2 as 0 or 1. Assuming the least significant bit of EB_1 or $EB_2 = 1$, we get:

$$R \cong PAVG(B_1,B_2) - (B_1 \wedge B_2)\ \&\ ONE. \tag{22}$$

This solution requires 6 instructions, and according to (16), it is close to the following:

$$R \cong PAVG(CLIP(B_1{-}ONE),B_2) \text{ or } R \cong PAVG(B_1,CLIP(B_2{-}ONE)). \tag{23}$$

This solution requires only 4 instructions, and produces a maximum error of ±1 in the final result for 12.5% of all possible values of $A_1,...,A_4$ between [0,255]. The error never exceeds ±1. We get a computational efficiency of 19:4, nearly 5 times speedup.

B. Type 2, Filter 2: $R = (A_1 + A_2 + A_3 + A_4 + ONE) >> 2$

i. Efficient SIMD Solution

[47] As seen in Section 3.A, this filter can be implemented by conventional SIMD methods in 19 instructions. For efficient implementation, we simplify as follows:

$$R = (((A_1{+}A_2{+}ONE){>>}1){+}((A_3{+}A_4){>>}1){+}E){>>}1 = (B_1{+}B_2{+}(E{-}(EB_2\&ONE))) >> 1. \tag{24}$$

Here EB_2 is due to the correction term in (12), and E is the correction term in (15) as:

$$\begin{aligned} E &= ODD(A_1{+}A_2{+}ONE)\ \&\ ODD(A_3{+}A_4) = EVEN(A_1{+}A_2)\ \&\ ODD(A_3{+}A_4) \\ &= {\sim}(A_1 \wedge A_2)\ \&\ (A_3 \wedge A_4)\ \&\ ONE = ({\sim}EB_1\ \&\ EB_2)\ \&\ ONE. \end{aligned} \tag{25}$$

We note that $E_T = (E - (EB_2\&ONE)) \in \{0,-1\}$. From (24), (11), and (12) we obtain:

$$R = \begin{cases} PAVG(B_1,B_2) - (B_1 \wedge B_2)\ \&\ ONE & \text{when } E_T = 0 \\ PAVG(B_1,B_2) - ONE & \text{when } E_T = -1 \end{cases}. \tag{26}$$

Note that $(E-(EB_2\&ONE)) = -1$ when $(A_1 \wedge A_2)\ \&\ (A_3 \wedge A_4)\ \&\ ONE = 1$. We simplify (26) as:

$$R = PAVG(B_1,B_2) - ((B_1 \wedge B_2)\ |\ ((A_1 \wedge A_2)\ \&\ (A_3 \wedge A_4)))\ \&\ ONE. \tag{27}$$

The solution in (27) requires 10 instructions, an approximate 19:10 (nearly 2 times) speedup.

ii. Approximate SIMD Solution

[48] We can obtain four approximations of (27) by assuming the least significant bit of EB_1 or EB_2 as 0 or 1. A good approximate solution is with the assumption that the least significant bit of EB_2=0, which gives us the same solutions as (22) and (23), which require 4 instructions and has a maximum error of ±1 for 12.5% of all possible values of $A_1,\ldots, A_4 \in [0,255]$. We get a computational advantage of 19:4.

C. Type 2, Filter 3: $R = (A_1 + A_2 + A_3 + A_4) >> 2$

i. Efficient SIMD Solution

[49] The filter can be implemented by conventional SIMD methods in 17 instructions. For efficient implementation, we simplify as follows:

$$R = (((A_1+A_2)>>1)+((A_3+A_4)>>1)+E)>>1 = (B_1+B_2+(E-(EB_1+EB_2)\&ONE))>>1. \tag{28}$$

Here EB_1 and EB_2 are due to the correction term in (12), and E is the correction term in (15) as:

$$E=ODD(A_1+A_2) \ \& \ ODD(A_3+A_4) = (A_1\hat{}A_2) \ \& \ (A_3\hat{}A_4) \ \& \ ONE = EB_1 \ \& \ EB_2 \ \& \ ONE. \tag{29}$$

We note that E_T=$(E-(EB_1+EB_2)\&ONE) \in \{0,-1\}$, and R is same as (26). Note that $E_T = -1$ when $(A_1\hat{}A_2) \mid (A_3\hat{}A_4) \ \& \ ONE = 1$. We simplify (26) as:

$$R = PAVG(B_1,B_2) - ((B_1\hat{}B_2) \mid (A_1\hat{}A_2) \mid (A_3\hat{}A_4)) \ \& \ ONE. \tag{30}$$

The solution in (30) requires 10 instructions. We have an approximate 17:10 speedup by using (30).

ii. Approximate SIMD Solution

[50] We can obtain four approximations of (30) by assuming the least significant bit of EB_1 or EB_2 as 0 or 1. A good approximate solution is with the assumptions that the least significant bits of EB_1 or EB_2 =1, which gives us:

$$R \cong CLIP(PAVG(B_1,B_2) - ONE). \tag{31}$$

This solution requires 4 instructions and has a maximum error of ±1 for 12.5% of all possible values of $A_1,\ldots,A_4 \in [0,255]$. We have a computational advantage of 17:4, approx. 4 times.

D. Type 2, Special Filter 1: $R = (2A_1 + A_3 + A_4 + 2{*}ONE) >> 2$

[51] This filter is same as Filter 1 with $A_1 = A_2$. It can be implemented by conventional SIMD methods in, e.g., 17 instructions. This type of filter is used extensively in, for example, standards proposed by the Joint Video Team (JVT) as, for example, in **[CHANGE THIS – CHANCHAL TO UPDATE TO MORE CURRENT REFERENCE →** ISO/IEC MPEG and ITU-T VCEG, Geneva, Switzerland, Oct., 02; entitled "Editor's Proposed Draft Text Modifications for Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC), Geneva modifications, draft 26.and other coding schemes. **← END REFERENCE]**

[52] We can simplify (21) as:

$$R = PAVG(A_1,B_2) - (A_1 \wedge B_2)\ \&\ (A_3 \wedge A_4)\ \&\ ONE. \tag{32}$$

This solution requires 7 instructions. One can verify that (32) is close to the following:

$$R \cong PAVG(A_1,PAVG(CLIP(A_3-ONE),A_4)), \tag{33}$$

which requires only 3 instructions instead of 17 instructions by conventional SIMD methods, a nearly 6 times speedup. However, (33) produces an error of ±1 for a very small 0.1% of all possible values of $A_1,\ldots,A_4 \in [0,255]$.

E. Type 2, Summary of Results

[53] Table II below summarizes the instructions required to compute each filter (given sufficient memory) by the efficient and conventional SIMD methods. For the Efficient case, we give the instructions required for the exact and approximate solutions.

Summary of Results for Type 2 FIR Filters.

Type 2 Filters	**Conventional Method**	**Efficient Method**		**Speedup**	
		Exact	**Approx.**	**Exact**	**Approx.**

$(A_1 + A_2 + A_3 + A_4 + 2*ONE) >> 2$	19	10	4	1.9	4.75
$(A_1 + A_2 + A_3 + A_4 + ONE) >> 2$	19	10	4	1.9	4.75
$(A_1 + A_2 + A_3 + A_4) >> 2$	17	10	4	1.7	4.25
$(2A_1 + A_2 + A_3 + 2*ONE) >> 2$	17	7	3	2.4	5.67

TABLE II

[54] The shaded areas show significant improvements in efficiency due to the analyses developed here.

3. Type 3 FIR Filters

[55] There are 5 different Type 3 FIR filters depending on the 5 choices of c in (8). We define the following packed 64-bit registers, each containing 8 data elements of one byte each:

$$B_1 = PAVG(A_1,A_2),\ B_2 = PAVG(A_3,A_4),\ B_3 = PAVG(A_5,A_6),\ B_4 = PAVG(A_7,A_8),$$
$$C_1 = PAVG(B_1,B_2),\ C_2 = PAVG(B_3,B_4),$$
$$EB_1 = (A_1 \hat{} A_2),\ EB_2 = (A_3 \hat{} A_4),\ EB_3 = (A_5 \hat{} A_6),\ EB_4 = (A_7 \hat{} A_8),$$
$$EC_1 = (B_1 \hat{} B_2),\ EC_2 = (B_3 \hat{} B_4). \quad (34)$$

A. Type 3, Filter 1: R = (A$_1$ + A$_2$ + A$_3$ + A$_4$ + A$_5$ + A$_6$ + A$_7$ + A$_8$ + 4*ONE) >> 3

i. Conventional SIMD Solution

[56] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (8 Instructions) A_{iL} = Unpack Low 4 Bytes of A_i, for $i = 1,\ldots,8$,
2. (8 Instructions) A_{iH} = Unpack High 4 Bytes of A_i, for $i = 1,\ldots,8$,
3. (9 Instructions) Add and Shift lower 4 words of $A_1,\ldots,A_8$ to obtain lower 4 words of R_L as:

   $R_L = (A_{1L} + A_{2L} + A_{3L} + A_{4L} + A_{5L} + A_{6L} + A_{7L} + A_{8L} + 4*ONE_4) >> 3,$
4. (9 Instructions) Add and Shift higher 4 words of $A_1,\ldots,A_8$ to obtain higher 4 words of R_H as:

$R_H = (A_{1H} + A_{2H} + A_{3H} + A_{4H} + A_{5H} + A_{6H} + A_{7H} + A_{8H} + 4{*}ONE_4) >> 3,$

5. (1 Instruction) Pack R_H and R_L into final register R.

We require 35 instructions to compute this filter by conventional SIMD methods.

ii. New SIMD Solution

[57] In order to implement this filter without unpacking, we simplify it as follows:

$$R = (((A_1 + A_2 + A_3 + A_4 + 2{*}ONE) >> 2) + ((A_5 + A_6 + A_7 + A_8 + 2{*}\mathrm{ONE}) >> 2) + E) >> 1\,, \quad (35)$$

where E is the error/correction term that is necessary for dividing up the expression into two parts each containing a Type 2, Filter 1 (21). Note that according to (21) and the registers in (34), we have:

$$(A_1 + A_2 + A_3 + A_4 + 2{*}ONE) >> 2 = C_1 - (E_1 \,\&\, ONE),$$
$$(A_5 + A_6 + A_7 + A_8 + 2{*}ONE) >> 2 = C_2 - (E_2 \,\&\, ONE), \quad (36)$$

where $E_1 = EC_1 \,\&\, (EB_1 \mid EB_2)$ and $E_2 = EC_2 \,\&\, (EB_3 \mid EB_4)$ are error/correction terms obtained in (21). We can simplify (35) as:

$$R = (C_1 + C_2 + (E - E_1 - E_2)\&ONE) >> 1. \quad (37)$$

We now find the expression for E in terms of the packed 64-bit registers in (34). The simplification in (35), amounts to the following:

$$(P+Q) >> 3 = ((P >> 2) + (Q >> 2) + E) >> 1, \quad (38)$$

where P and Q are unsigned integers. Let p_0 be the least significant bit of P and p_1 the next significant bit of P. Similarly, let q_0 be the least significant bit of Q and q_1 the next significant bit of Q. The simplification in (38) results in an error E when the last 2 bits of P and Q add up to a number ≥ 4. The condition that determines this error E is:

$$(p_1 \,\&\, q_1) \mid (p_0 \,\&\, q_0 \,\&\, (p_1 \mid q_1)).$$

We can prove that p_1, p_0, q_1, q_0 can be expressed in terms of the registers in (34) as the least significant bits of the following packed 64-bit registers respectively:

$$P_1 = (EC_1 \wedge \sim(EB_1 \mid EB_2)),$$
$$P_0 = (EB_1 \wedge EB_2),$$
$$Q_1 = (EC_2 \wedge \sim(EB_3 \mid EB_4)),$$

$$Q_0 = (EB_3 \wedge EB_4), \tag{39}$$

From (39), we can express E as the least significant bit of:

$$E = (P_1 \,\&\, Q_1) \,|\, (P_0 \,\&\, Q_0 \,\&\, (P_1 \,|\, Q_1)). \tag{40}$$

We note that $E \in \{0, 1\}$, and $E_T = (E - E_1 - E_2) \in \{-1, 0, 1\}$. From (37), we have:

$$R = \begin{cases} PAVG(C_1,C_2) - ONE & \text{when } E_T = -1 \\ PAVG(C_1,C_2) - (C_1 \wedge C_2) \,\&\, ONE & \text{when } E_T = 0 \\ PAVG(C_1,C_2) & \text{when } E_T = 1 \end{cases}. \tag{41}$$

We simplify (41) as:

$$R = PAVG(C_1,C_2) - ((C_1 \wedge C_2) \,|\, (E_1 \wedge E_2 \wedge E)) \,\&\, (E_1 \,|\, E_2 \,|\, {\sim}E) \,\&\, ONE, \tag{42}$$

where $E_1 = EC_1 \,\&\, (EB_1 \,|\, EB_2)$ and $E_2 = EC_2 \,\&\, (EB_3 \,|\, EB_4)$. We can further simplify (42) as an expression in terms of EC_1, EC_2, EB_1, EB_2, EB_3, and EB_4 so that we can skip the computations of E_1, E_2, and E as follows:

$$U = EC_1 \,|\, EC_2,$$
$$V = EB_1 \,|\, EB_2,$$
$$W = EB_3 \,|\, EB_4,$$
$$X = V \,|\, W,$$
$$Y = U \,|\, X,$$
$$Z = (EC_1 \,\&\, EC_2 \,\&\, X),$$
$$T = U \,\&\, V \,\&\, W \,\&\, ((EB_1 \,\&\, EB_2) \,|\, (EB_3 \,\&\, EB_4)),$$
$$R = PAVG(C_1,C_2) - ((C_1 \wedge C_2) \,|\, Z \,|\, T) \,\&\, Y \,\&\, ONE, \tag{43}$$

[58] The solution in (43) is shown in pseudo-code in Table III, below. Any suitable language, coding technique, circuitry or combination of hardware and software can be used to achieve the functionality shown in the pseudo-code presented herein. The approach of Table III uses 32 instructions as compared to the conventional 35 instructions. The count of 32 instructions is obtained by counting each logical and arithmetic operation of (43) along with those of (34). Other instruction counts in this application are obtained, similarly. Clearly, the approach of Table III is not as efficient as the Type 2 algorithms. However, there are at least 2 benefits of this approach:

(1) We can systematically arrive at approximate solutions by making assumptions on the error/correction terms EC_1, EC_2, EB_1, EB_2, EB_3, and EB_4 (see Section 3.A.iii).

(2) In special cases, where various A_i's are same, we can simplify the computation considerably and obtain efficient exact and approximate solutions (see Sections 3.F – 3.H).

```
#define P(a,b)    (((a) + (b) + 1) >> 1)
b1 = P(a01,a02);
b2 = P(a03,a04);
b3 = P(a05,a06);
b4 = P(a07,a08);
c1 = P(b1 ,b2 );
c2 = P(b3 ,b4 );
d  = P(c1 ,c2 );
eb1 = a01 ^ a02;
eb2 = a03 ^ a04;
eb3 = a05 ^ a06;
eb4 = a07 ^ a08;
ec1 = b1  ^ b2;
ec2 = b3  ^ b4;
ed  = c1  ^ c2;
u  = ec1 | ec2;
v  = eb1 | eb2;
w  = eb3 | eb4;
x  = v | w;
y  = u | x;
z  = ec1 & ec2 & x;
t  = u & v & w & ((eb1 & eb2) | (eb3 & eb4));
e  = ((ed & y) | z | t) & 0x01; // Exact solution
x1  = CLIP(d - e);
```

TABLE III

iii. Approximate SIMD Solution

[59] We have many approximate solutions by assuming the least significant bit of EB_1, EB_2, EB_3, EB_4, EC_1, or EC_2 as 0 or 1. With the assumption that the least significant bit of $EB_1 = 1$, and $EB_2 = EB_3 = EB_4 = 0$, we get from (43):

$$R = PAVG(C_1,C_2) - ((C_1 \wedge C_2) \mid (EC_1 \ \& \ EC_2)) \ \& \ ONE. \qquad (44)$$

[60] An example pseudo-code implementation of this solution is shown in Table IV, below. This approach uses 14 instructions, and produces a maximum error of ±1 in the final result for 9.38% of all possible values of $A_1,\ldots,A_8$ between [0,255]. The error never exceeds ±1. This solution with 14 instructions, and a maximum error of ±1 for less than 1/10th of the data is acceptable in many applications like post-processing, where a difference of 1 gray value in the displayed frame is imperceptible to most of us. Yet, we receive a computational advantage of 35:14.

[61] The second approximate solution makes the assumption $EB_1 = 0$, and $EB_2 = 1$. It produces the following solution:

$$T = (EC_1 \mid EC_2)\ \&\ (EB_3 \mid EB_4)\ \&\ EB_3\ \&\ EB_4,$$
$$R = PAVG(C_1,C_2) - ((C_1 \wedge C_2) \mid (EC_1\ \&\ EC_2) \mid T)\ \&\ ONE, \qquad (45)$$

This solution requires 22 instructions, and produces a maximum error of ±1 in the final result for 6.25% of all possible values of $A_1,\ldots,A_8$ between [0,255].

```
#define P(a,b)    (((a) + (b) + 1) >> 1)
b1 = P(a01,a02);
b2 = P(a03,a04);
b3 = P(a05,a06);
b4 = P(a07,a08);
c1 = P(b1 ,b2 );
c2 = P(b3 ,b4 );
d  = P(c1 ,c2 );
ec1 = b1  ^ b2;
ec2 = b3  ^ b4;
ed  = c1  ^ c2;
e   = (ed | (ec1 & ec2)) & 0x01; //approx = 9.375%
x1  = CLIP(d - e);
```

TABLE IV

B. Type 3, Filter 2: $R = (A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8 + 3{*}ONE) >> 3$

[62] We require 35 instructions to compute this filter by conventional SIMD methods. For the new SIMD solution, we write the filter as:

[63] $R = (((A_1{+}A_2{+}A_3{+}A_4{+}2{*}ONE){>>}2) + ((A_5{+}A_6{+}A_7{+}A_8{+}ONE){>>}2) + E) >> 1,$

where E is the error/correction term that is necessary for dividing up the expression into two parts each containing a Type 2 Filter. We have:

$$R = (C_1 + C_2 + (E - E_1 - E_2)\&ONE) >> 1,$$
$$(A_1{+}A_2{+}A_3{+}A_4{+}2{*}ONE){>>}2 = C_1 - (E_1 \ \& \ ONE),$$
$$(A_5{+}A_6{+}A_7{+}A_8{+}ONE){>>}2 = C_2 - (E_2 \ \& \ ONE),$$
$$E_1 = EC_1 \ \& \ (EB_1 \mid EB_2),\ E_2 = EC_2 \mid (EB_3 \ \& \ EB_4),$$
$$P_1 = (EC_1 \wedge {\sim}(EB_1 \mid EB_2)),\ P_0 = (EB_1 \wedge EB_2),$$
$$Q_1 = (EC_2 \wedge (EB_3 \ \& \ EB_4)),\ Q_0 = {\sim}(EB_3 \wedge EB_4),$$
$$E = (P_1 \ \& \ Q_1) \mid (P_0 \ \& \ Q_0 \ \& \ (P_1 \mid Q_1)).$$

Here E_1 and E_2 are error/correction terms obtained from (21) and (27) respectively. Defining $E_T = (E - E_1 - E_2) \in \{-2, -1, 0\}$, we have:

$$R = \begin{cases} PAVG(C_1,C_2) - (C_1 \wedge C_2)\ \&\ ONE & \text{when } E_T = 0 \\ PAVG(C_1,C_2) - ONE & \text{when } E_T = -1\,. \\ PAVG(C_1,C_2) - ONE - (C_1 \wedge C_2)\ \&\ ONE & \text{when } E_T = -2 \end{cases} \qquad (46)$$

We simplify (46) as:

$$S = (E_1 \wedge E_2 \wedge E),$$
$$R = PAVG(C_1,C_2) - ((E_1 \ \& \ E_2 \ \& \ {\sim}E) \mid S) \ \& \ ONE - ((C_1 \wedge C_2) \ \& \ {\sim}S) \ \& \ ONE, \qquad (47)$$

This solution can be further simplified as:

$$P = EB_3 \ \& \ EB_4,$$
$$U = EB_1 \ \& \ EB_2 \ \& \ P,$$
$$V = EC_1 \ \& \ EC_2,$$
$$W = EB_3 \mid EB_4,$$
$$X = (EC_1 \mid EC_2) \ \& \ ((EB_1 \ \& \ (EB_2 \mid W)) \mid (EB_2 \ \& \ W) \mid P),$$
$$Y = (X \mid V \mid U),$$
$$ED = (C_1 \wedge C_2),$$
$$R = PAVG(C_1,C_2) - ((ED \mid Y) \ \& \ ONE) - (U \ \& \ V \ \& \ ED \ \& \ ONE). \qquad (48)$$

The solution in (47) requires 35 instructions, same as the conventional 35 instructions.

[64] An approximate solution of (47) can be obtained with the assumption that the least significant bit of $EB_1 = EB_2 = 1$, and $EB_3 = EB_4 = 0$ is:

$$R = PAVG(C_1,C_2) - ((C_1 \wedge C_2) \mid EC_1 \mid EC_2) \,\&\, ONE. \tag{49}$$

[65] This solution requires 14 instructions, and produces a maximum error of ±1 in the final result for 9.38% of all possible values of $A_1,\ldots,A_8$ between [0,255]. We receive a computational advantage of 35:14.

[66] The second approximate solution makes the assumption EB1 = 1, and EB2 = 0. It produces the following solution:

$$Y = ((EC_1 \mid EC_2) \,\&\, (EB_3 \mid EB_4)) \mid (EC_1 \,\&\, EC_2),$$
$$R = PAVG(C_1,C_2) - ((C_1 \wedge C_2) \mid Y) \,\&\, ONE. \tag{50}$$

This solution requires 20 instructions, and produces a maximum error of ±1 in the final result for 6.25% of all possible values of $A_1,\ldots,A_8$ between [0,255].

C. Type 3, Filter 3: R = (A$_1$ + A$_2$ + A$_3$ + A$_4$ + A$_5$ + A$_6$ + A$_7$ + A$_8$ + 2*ONE) >> 3

[67] We require 35 instructions to compute this filter by conventional SIMD methods. For the new SIMD solution, we write the filter as:

$$R = (((A_1{+}A_2{+}A_3{+}A_4{+}ONE){>>}2) + ((A_5{+}A_6{+}A_7{+}A_8{+}ONE){>>}2) + E) >> 1,$$

where E is the error/correction term that is necessary for dividing up the expression into two parts each containing a Type 2, Filter 2 (27). We have:

[68] $R = (C_1 + C_2 + (E - E_1 - E_2)\&ONE) >> 1,$

$$(A_1{+}A_2{+}A_3{+}A_4{+}ONE){>>}2 = C_1 - (E_1 \,\&\, ONE),$$
$$(A_5{+}A_6{+}A_7{+}A_8{+}ONE){>>}2 = C_2 - (E_2 \,\&\, ONE),$$
$$E_1 = EC_1 \mid (EB_1 \,\&\, EB_2),\ E_2 = EC_2 \mid (EB_3 \,\&\, EB_4),$$
$$P_1 = (EC_1 \wedge (EB_1 \,\&\, EB_2)),\ P_0 = {\sim}(EB_1 \wedge EB_2),$$
$$Q_1 = (EC_2 \wedge (EB_3 \,\&\, EB_4)),\ Q_0 = {\sim}(EB_3 \wedge EB_4),$$
$$E = (P_1 \,\&\, Q_1) \mid (P_0 \,\&\, Q_0 \,\&\, (P_1 \mid Q_1)).$$

Here E_1 and E_2 are error/correction terms obtained from (27). Defining $E_T = (E - E_1 - E_2) \in \{-2, -1, 0\}$, we have the same expression for R as in (46), which we simplify as:

$$S = (E_1 \wedge E_2),$$
$$R = PAVG(C_1,C_2) - ({\sim}(E \,\&\, S) \,\&\, (E_1 \mid E_2 \mid E)) \,\&\, ONE - ((C_1 \wedge C_2) \,\&\, {\sim}(S \wedge E)) \,\&\, ONE. \tag{51}$$

[69] This solution can be further simplified as:

$$P = EB_1 \mid EB_4,$$

$$Q = EB_3 \mid EB_2,$$

$$U = (EB_2 \,\&\, EB_3 \,\&\, P) \mid (EB_4 \,\&\, EB_1 \,\&\, Q),$$

$$V = EC_1 \,\&\, EC_2,$$

$$W = P \mid Q,$$

$$ED = (C_1 \wedge C_2),$$

$$R = PAVG(C_1,C_2) - (ED \mid U \mid V \mid ((EC_1 \mid EC_2) \,\&\, W)) \,\&\, ONE - ED \,\&\, U \,\&\, V \,\&\, ONE. \quad (52)$$

The solution in (52) requires 34 instructions, close to the conventional 35 instructions.

[70] The approximate solution requires the assumption that the least significant bit of $EB_1 = 1$, and $EB_2 = EB_3 = EB_4 = 0$ is:

$$R = PAVG(C_1,C_2) - ((C_1 \wedge C_2) \mid EC_1 \mid EC_2) \;\&\, ONE. \quad (53)$$

This solution requires 14 instructions, and produces a maximum error of ±1 in the final result for 9.38% of all possible values of $A_1,\ldots,A_8$ between [0,255]. We receive a computational advantage of 35:14.

[71] The second approximate solution makes the assumption $EB_1 = 1$, and $EB_2 = 0$. It produces the following solution:

$$U = EB_3 \,\&\, EB_4,$$

$$ED = (C_1 \wedge C_2),$$

$$R = PAVG(C_1,C_2) - (ED \mid U \mid EC_1 \mid EC_2) \,\&\, ONE - ED \,\&\, U \,\&\, EC_1 \,\&\, EC_2 \,\&\, ONE. \quad (54)$$

This solution requires 23 instructions, and produces a maximum error of ±1 in the final result for 6.25% of all possible values of $A_1,\ldots,A_8$ between [0,255].

D. Type 3, Filter 4: $R = (A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8 + ONE) >> 3$

[72] We require 35 instructions to compute this filter by conventional SIMD methods. For the new SIMD solution, we write the filter as:

$$R = (((A_1+A_2+A_3+A_4+ONE)>>2) + ((A_5+A_6+A_7+A_8)>>2) + E) >> 1,$$

where E is the error/correction term that is necessary for dividing up the expression into two parts each containing a Type 2, Filter. We have:

$$R = (C_1 + C_2 + (E - E_1 - E_2)\&ONE) >> 1,$$
$$(A_1{+}A_2{+}A_3{+}A_4{+}ONE){>>}2 = C_1 - (E_1 \ \& \ ONE),$$
$$(A_5{+}A_6{+}A_7{+}A_8){>>}2 = C_2 - (E_2 \ \& \ ONE),$$
$$E_1 = EC_1 \mid (EB_1 \ \& \ EB_2), E_2 = EC_2 \mid EB_3 \mid EB_4,$$
$$P_1 = (EC_1 \wedge (EB_1 \ \& \ EB_2)), P_0 = \sim(EB_1 \wedge EB_2),$$
$$Q_1 = (EC_2 \wedge (EB_3 \mid EB_4)), Q_0 = (EB_3 \wedge EB_4),$$
$$E = (P_1 \ \& \ Q_1) \mid (P_0 \ \& \ Q_0 \ \& \ (P_1 \mid Q_1)).$$

Here E_1 and E_2 are error/correction terms obtained from (27) and (30) respectively. Defining $ET = (E - E_1 - E_2) \in \{-2, -1, 0\}$, we have the same expression for R as in (46), which we simplify as:

$$S = (E_2 \wedge E),$$
$$R = PAVG(C_1,C_2) - (E_1 \mid S) \ \& \ ONE - ((C_1 \wedge C_2) \ \& \sim(E_1 \wedge S)) \ \& \ ONE, \tag{55}$$

This solution can be further simplified as:

$$P = EB_3 \ \& \ EB_4,$$
$$Q = EB_3 \mid EB_4,$$
$$U = (EB_1 \ \& \ (EB_2 \mid Q)) \mid (EB_2 \ \& \ Q) \mid P,$$
$$V = EB_1 \ \& \ EB_2 \ \& \ P,$$
$$W = EC_1 \mid EC_2,$$
$$Z = (EC_1 \ \& \ EC_2 \ \& \ U),$$
$$ED = (C_1 \wedge C_2),$$
$$R = PAVG(C_1,C_2) - (ED \mid U \mid W) \ \& \ ONE - ED \ \& \ ((W \ \& \ V) \mid Z) \ \& \ ONE. \tag{56}$$

The solution in (56) requires 35 instructions, same as the conventional 35 instructions. The approximate solution requires the assumption that the least significant bit of EB_1 = $EB_2 = 0$, and $EB_3 = EB_4 = 1$ is:

$$R = PAVG(C_1,C_2) - ONE - ((C_1 \wedge C_2) \ \& \ EC_1 \ \& \ EC_2) \ \& \ ONE. \tag{57}$$

This solution requires 15 instructions, and produces a maximum error of ±1 in the final result for 9.38% of all possible values of $A_1,\ldots,A_8$ between [0,255]. We receive a computational advantage of 35:15.

[73] The second approximate solution makes the assumption $EB_1 = 1$, and $EB_2 = 0$. It produces the following solution:

$$Q = EB_3 \mid EB_4,$$

$$Z = (EC_1 \mathbin{\&} EC_2 \mathbin{\&} Q),$$

$$ED = (C_1 \wedge C_2),$$

$$R = PAVG(C_1,C_2) - ((ED \mid Q \mid EC_1 \mid EC_2) \mathbin{\&} ONE) - (ED \mathbin{\&} Z \mathbin{\&} ONE). \qquad (58)$$

This solution requires 23 instructions, and produces a maximum error of ±1 in the final result for 6.25% of all possible values of $A_1,\ldots,A_8$ between [0,255].

E. Type 3, Filter 5: $R = (A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8) >> 3$

[74] We require 33 instructions to compute this filter by conventional SIMD methods. For the new SIMD solution, we write the filter as:

$$R = (((A_1+A_2+A_3+A_4)>>2) + ((A_5+A_6+A_7+A_8)>>2) + E) >> 1,$$

where E is the error/correction term that is necessary for dividing up the expression into two parts each containing a Type 2, Filter (30). We have:

$$R = (C_1 + C_2 + (E - E_1 - E_2)\mathbin{\&}ONE) >> 1,$$

$$(A_1+A_2+A_3+A_4)>>2 = C_1 - (E_1 \mathbin{\&} ONE),$$

$$(A_5+A_6+A_7+A_8)>>2 = C_2 - (E_2 \mathbin{\&} ONE),$$

$$E_1 = EC_1 \mid EB_1 \mid EB_2,\ E_2 = EC_2 \mid EB_3 \mid EB_4,$$

$$P_1 = (EC_1 \wedge (EB_1 \mid EB_2)),\ P_0 = (EB_1 \wedge EB_2),$$

$$Q_1 = (EC_2 \wedge (EB_3 \mid EB_4)),\ Q_0 = (EB_3 \wedge EB_4),$$

$$E = (P_1 \mathbin{\&} Q_1) \mid (P_0 \mathbin{\&} Q_0 \mathbin{\&} (P_1 \mid Q_1)).$$

Here E_1 and E_2 are error/correction terms obtained from (27) and (30) respectively. Defining $ET = (E - E_1 - E_2) \in \{-2, -1, 0\}$, we have the same expression for R as in (46), which we simplify as:

$$R = PAVG(C_1,C_2) - (E_1 \mid E_2) \mathbin{\&} ONE - ((C_1 \wedge C_2) \mathbin{\&} \sim(E_1 \wedge E_2 \wedge E)) \mathbin{\&} ONE, \qquad (59)$$

This solution can be simplified as:

$$P = EB_1 \mid EB_4,$$

$$Q = EB_3 \mid EB_2,$$

$$U = P \mid Q,$$

$$V = (EB_2 \mathbin{\&} EB_3 \mathbin{\&} P) \mid (EB_4 \mathbin{\&} EB_1 \mathbin{\&} Q),$$

$$W = EC_1 \mid EC_2,$$

$$Z = (EC_1 \mathbin{\&} EC_2 \mathbin{\&} U),$$

$$ED = (C_1 \wedge C_2),$$

$$R = PAVG(C_1,C_2) - ((ED \mid U \mid W) \mathbin{\&} ONE) - (ED \mathbin{\&} ((W \mathbin{\&} V) \mid Z) \mathbin{\&} ONE). \qquad (60)$$

The solution in (59) requires 34 instructions, close to the conventional 35 instructions.

[75] The approximate solution requires the assumption that the least significant bit of $EB_1 = 1$, and $EB_2 = EB_3 = EB_4 = 0$ is:

$$R = PAVG(C_1,C_2) - ONE - (C_1 \wedge C_2) \mathbin{\&} EC_1 \mathbin{\&} EC_2 \mathbin{\&} ONE. \qquad (61)$$

This solution requires 15 instructions, and produces a maximum error of ±1 in the final result for 9.38% of all possible values of $A_1,\ldots,A_8$ between [0,255]. We receive a computational advantage of 33:15.

[76] The second approximate solution makes the assumption EB1 = 1, and EB2 = 0. It produces the following solution:

$$W = EC_1 \mid EC_2,$$

$$R = PAVG(C_1,C_2) - ONE - ((C_1 \wedge C_2) \mathbin{\&} ((W \mathbin{\&} EB_3 \mathbin{\&} EB_4) \mid (EC_1 \mathbin{\&} EC_2)) \mathbin{\&} ONE). \qquad (62)$$

This solution requires 21 instructions, and produces a maximum error of ±1 in the final result for 6.25% of all possible values of $A_1,\ldots,A_8$ between [0,255].

F. Type 3, Special Filter 1: R = ($A_1 + 2A_3 + 2A_5 + 2A_7 + A_2$ + 4*ONE) >> 3

[77] This filter is an important loop filter for de-blocking in JVT video compression standards.

i. Conventional SIMD Solution

[78] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (5 Instructions) $A_{i\text{L}}$ = Unpack Low 4 Bytes of A_i, for $i \in \{1,2,3,5,7\}$,
2. (5 Instructions) $A_{i\text{H}}$ = Unpack High 4 Bytes of A_i, for $i \in \{1,2,3,5,7\}$,
3. (9 Instructions) Add and Shift lower 4 words of $A_1,\ldots,A_5$ to obtain lower 4 words of R_L as:

   $R_\text{L} = (A_{1\text{L}} + 2A_{3\text{L}} + 2A_{5\text{L}} + 2A_{7\text{L}} + A_{2\text{L}} + 4{*}ONE_4) >> 3,$
4. (9 Instructions) Add and Shift higher 4 words of $A_1,\ldots,A_5$ to obtain higher 4 words of R_H as:

   $R_\text{H} = (A_{1\text{H}} + 2A_{3\text{H}} + 2A_{5\text{H}} + 2A_{7\text{H}} + A_{2\text{H}} + 4{*}ONE_4) >> 3,$
5. (1 Instruction) Pack R_H and R_L into final register R.

We require 29 instructions to compute this filter by conventional SIMD methods.

ii. Efficient SIMD Solution

[79] From (34), we get the:

$$\begin{gathered}B_1 = PAVG(A_1,A_2),\ B_2 = A_3,\ B_3 = A_5,\ B_4 = A_7,\\ C_1 = PAVG(B_1,A_3),\ C_2 = PAVG(A_5,A_7),\\ EB_1 = (A_1\wedge A_2),\ EB_2 = 0,\ EB_3 = 0,\ EB_4 = 0,\\ EC_1 = (B_1\wedge A_3),\ EC_2 = (A_5\wedge A_7).\end{gathered} \tag{63}$$

In (43) we get:

$$R = PAVG(C_1,C_2) - ((C_1\wedge C_2) \mid (EC_1 \mathbin{\&} EC_2 \mathbin{\&} EB_1)) \mathbin{\&} (EC_1 \mid EC_2 \mid EB_1) \mathbin{\&} ONE. \tag{64}$$

The solution in (64) requires 16 instructions with a computational benefit of 29:16.

iii. Approximate SIMD Solution

[80] The approximate solution with the assumptions that the least significant bit of EB_1 = EC_1 = 0, and EC_2 = 1 is:

$$R = PAVG(C_1,C_2) - (C_1\wedge C_2) \mathbin{\&} ONE. \tag{65}$$

It requires 7 instructions, and produces a maximum error of ±1 in the final result for 12.5% of all possible values of $A_1,\ldots,A_8$ between [0,255]. The computational advantage is 29:7 (approx. 4 times speedup).

G. Type 3, Special Filter 2: $R = (A_1 + A_2 + A_3 + 3A_4 + 2A_7 + 4*ONE) >> 3$

[81] This filter is also an important loop filter for de-blocking in the JVT video compression standard.

i. Conventional SIMD Solution

[82] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (5 Instructions) A_{iL} = Unpack Low 4 Bytes of A_i, for $i \in \{1,2,3,4,7\}$,
2. (5 Instructions) A_{iH} = Unpack High 4 Bytes of A_i, for $i \in \{1,2,3,4,7\}$,
3. (8 Instructions) Add and Shift lower 4 words of $A_1,\ldots,A_5$ to obtain lower 4 words of R_L as:

   $R_L = (A_{1L} + A_{2L} + A_{3L} + 3A_{4L} + 2A_{7L} + 4*ONE_4) >> 3,$
4. (8 Instructions) Add and Shift higher 4 words of $A_1,\ldots,A_5$ to obtain higher 4 words of R_H as:

   $R_H = (A_{1H} + A_{2H} + A_{3H} + 3A_{4H} + 2A_{7H} + 4*ONE_4) >> 3,$
5. (1 Instruction) Pack R_H and R_L into final register R.

[83] We require 27 instructions (including two multiplications by 3) to compute this filter by conventional SIMD methods.

ii. Efficient SIMD Solution

[84] From (34), we get the:

$$\begin{gathered} B_1 = PAVG(A_1,A_2),\ B_2 = PAVG(A_3,A_4),\ B_3 = A_4,\ B_4 = A_7, \\ C_1 = PAVG(B_1,B_2),\ C_2 = PAVG(A_4,A_7), \\ EB_1 = (A_1 \wedge A_2),\ EB_2 = (A_3 \wedge A_4),\ EB_3 = 0,\ EB_4 = 0, \\ EC_1 = (B_1 \wedge B_2),\ EC_2 = (A_4 \wedge A_7). \end{gathered} \tag{66}$$

In (43) we get:

$$S = (EB_1 \mid EB_2),$$

$$R = PAVG(C_1,C_2) - ((C_1 \text{^} C_2) \mid (EC_1 \,\&\, EC_2 \,\&\, S)) \,\&\, (EC_1 \mid EC_2 \mid S) \,\&\, ONE. \tag{67}$$

The solution in (65) requires 19 instructions with a computational benefit of 27:19.

iii. Approximate SIMD Solution

[85] The first approximate solution with the assumptions that the least significant bit of $EC_1 = EC_2 = 1$, and $EB_1 = EB_2 = 0$ is:

$$R = PAVG(C_1,C_2) - (C_1 \text{^} C_2) \,\&\, ONE. \tag{68}$$

It requires 8 instructions, and produces a maximum error of ±1 in the final result for 12.5% of all possible values of $A_1,\ldots,A_8$ between [0,255]. The computational advantage is 27:8.

The next approximate solution is with the assumption that the last bit of $EB_1 = EB_2 = 1$, which gives us:

$$R = PAVG(C_1,C_2) - ((C_1 \text{^} C_2) \mid (EC_1 \,\&\, EC_2)) \,\&\, ONE. \tag{69}$$

This solution requires 12 instructions and produces a maximum error of ±1 in the final result for 6.25% of all possible values of $A_1,\ldots, A_8$ between [0,255]. The computational advantage is 27:12.

H. Type 3, Special Filter 3: $R = (A_1 + A_2 + A_3 + 2A_4 + A_5 + A_6 + A_7 + 4{*}ONE) >> 3$

[86] This filter is used for de-blocking in post-processing.

i. Conventional SIMD Solution

[87] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (7 Instructions) A_{iL} = Unpack Low 4 Bytes of A_i, for $i \in \{1,2,3,4,5,67\}$,
2. (7 Instructions) A_{iH} = Unpack High 4 Bytes of A_i, for $i \in \{1,2,3,4,5,67\}$,
3. (9 Instructions) Add and Shift lower 4 words of $A_1,\ldots,A_5$ to obtain lower 4 words of R_L as:

   $R_L = (A_{1L} + A_{2L} + A_{3L} + 2A_{4L} + A_{5L} + A_{6L} + A_{7L} + 4{*}ONE_4) >> 3,$

4. (9 Instructions) Add and Shift higher 4 words of $A_1,\ldots,A_5$ to obtain higher 4 words of R_H as:

   $R_H = (A_{1H} + A_{2H} + A_{3H} + 2A_{4H} + A_{5H} + A_{6H} + A_{7H} + 4*ONE_4) >> 3,$

5. (1 Instruction) Pack R_H and R_L into final register R.

We require 33 instructions to compute this filter by conventional SIMD methods.

ii. Efficient SIMD Solution

[88] From (34), we get the:

$$B_1 = PAVG(A_1,A_2), B_2 = PAVG(A_3,A_5),\ B_3 = PAVG(A_6,A_7), B_4 = A_4,$$
$$C_1 = PAVG(B_1,B_2),\ C_2 = PAVG(B_3,A_4),$$
$$EB_1 = (A_1 \wedge A_2), EB_2 = (A_3 \wedge A_4), EB_3 = (A_6 \wedge A_7), EB_4 = 0,$$
$$EC_1 = (B_1 \wedge B_2), EC_2 = (B_3 \wedge A_4). \qquad (70)$$

In (43) we get:

$$U = EC_1 \mid EC_2,$$
$$V = EB_1 \mid EB_2,$$
$$X = V \mid EB_3,$$
$$Y = U \mid X,$$
$$Z = (EC_1 \;\&\; EC_2 \;\&\; X),$$
$$T = U \;\&\; V \;\&\; EB_1 \;\&\; EB_2 \;\&\; EB_3,$$
$$R = PAVG(C_1,C_2) - ((C_1 \wedge C_2) \mid Z \mid T) \;\&\; Y \;\&\; ONE, \qquad (71)$$

The solution in (71) requires 27 instructions with a computational benefit of 33:27.

iii. Approximate SIMD Solution

[89] We get an approximate solution with the assumptions that the least significant bit of $EB_2 = 1$, and $EB_2 = EB_3 = 0$ as:

$$R = PAVG(C_1,C_2) - ((C_1 \wedge C_2) \mid\ (EC_1 \;\&\; EC_2)) \;\&\; ONE. \qquad (72)$$

The solution in (72) requires 13 instructions, and produces a maximum error of ±1 in the final result for 6.25% of all possible values of $A_1,\ldots,A_7$ between [0,255]. The computational advantage is 33:13.

I. Type 3, Summary of Results

[90] Table V below summarizes the instructions required to compute each filter (given sufficient memory) by the efficient and conventional SIMD methods. For the Efficient case, we give the instructions required for the exact and approximate solutions.

Summary of Results for Type 3 FIR Filters.

Type 3 Filters	Conventional Method	Efficient Method		Speedup	
		Exact	Approx.	Exact	Approx.
$(A_1 + A_2 + \ldots + A_8 + 4{*}ONE) >> 3$	35	32	14	1.1	2.5
$(A_1 + A_2 + \ldots + A_8 + 3{*}ONE) >> 3$	35	35	14	1.0	2.5
$(A_1 + A_2 + \ldots + A_8 + 2{*}ONE) >> 3$	35	34	14	1.0	2.5
$(A_1 + A_2 + \ldots + A_8 + 1{*}ONE) >> 3$	35	35	15	1.0	2.3
$(A_1 + A_2 + \ldots + A_8) >> 3$	33	34	15	1.0	2.2
$(A_1{+}2A_3{+}2A_5{+}2A_7{+}A_2{+}4{*}ONE){>>}3$	29	16	7	1.8	4.1
$(A_1{+}A_2{+}A_3{+}3A_4{+}2A_7{+}4{*}ONE){>>}3$	27	19	8	1.4	3.4
$(A_1{+}A_2{+}A_3{+}2A_4{+}A_5{+}A_6{+}A_7{+}4{*}ONE){>>}3$	33	27	13	1.2	2.5

TABLE V

[91] The shaded areas show significant improvements in efficiency due to the analyses developed here.

4. Type 4 FIR Filters

[92] There are 9 different Type 4 FIR filters depending on the 9 choices of c in (9). For the sake of brevity, we shall only discuss the case of c=8. We define the following packed 64-bit registers, each containing 8 data elements of one byte each:

$$B_1 = PAVG(A_1,A_2),\ B_2 = PAVG(A_3,A_4),\ B_3 = PAVG(A_5,A_6),\ B_4 = PAVG(A_7,A_8),$$

$$B_5 = PAVG(A_9,A_{10}),\ B_6 = PAVG(A_{11},A_{12}),\ B_7 = PAVG(A_{13},A_{14}),\ B_8 = PAVG(A_{15},A_{16}),$$

$$C_1 = PAVG(B_1,B_2),\ C_2 = PAVG(B_3,B_4),\ C_3 = PAVG(B_5,B_6),\ C_4 = PAVG(B_7,B_8),$$

$$D_1 = PAVG(C_1,C_2),\ D_2 = PAVG(C_3,C_4),$$

$$EB_1 = (A_1 \wedge A_2),\ EB_2 = (A_3 \wedge A_4),\ EB_3 = (A_5 \wedge A_6),\ EB_4 = (A_7 \wedge A_8),$$
$$EB_5 = (A_9 \wedge A_{10}),\ EB_6 = (A_{11} \wedge A_{12}),\ EB_7 = (A_{13} \wedge A_{14}),\ EB_8 = (A_{15} \wedge A_{16}),$$
$$EC_1 = (B_1 \wedge B_2),\ EC_2 = (B_3 \wedge B_4),\ EC_3 = (B_5 \wedge B_6),\ EC_4 = (B_7 \wedge B_8),$$
$$ED_1 = (C_1 \wedge C_2),\ ED_2 = (C_3 \wedge C_4),$$

$$E_1 = EC_1 \,\&\, (EB_1 \mid EB_2),\ E_2 = EC_2 \,\&\, (EB_3 \mid EB_4),$$
$$E_3 = EC_3 \,\&\, (EB_5 \mid EB_6),\ E_4 = EC_4 \,\&\, (EB_7 \mid EB_8),$$

$$P_1 = (EC_1 \wedge \sim(EB_1 \mid EB_2)),\ \ P_0 = (EB_1 \wedge EB_2),$$
$$Q_1 = (EC_2 \wedge \sim(EB_3 \mid EB_4)),\ Q_0 = (EB_3 \wedge EB_4),$$
$$ER_1 = (P_1 \,\&\, Q_1) \mid ((P_1 \mid Q_1) \,\&\, P_0 \,\&\, Q_0),$$

$$R_1 = (EC_3 \wedge \sim(EB_5 \mid EB_6)),\ \ R_0 = (EB_5 \wedge EB_6),$$
$$S_1 = (EC_4 \wedge \sim(EB_7 \mid EB_8)),\ S_0 = (EB_7 \wedge EB_8),$$
$$ER_2 = (R_1 \,\&\, S_1) \mid ((R_1 \mid S_1) \,\&\, R_0 \,\&\, S_0),$$

$$U_2 = ED_1 \wedge E_1 \wedge E_2 \wedge ((P_1 \,\&\, Q_1) \mid ((P_1 \mid Q_1) \,\&\, P_0 \,\&\, Q_0)),$$
$$U_1 = P_1 \wedge Q_1 \wedge (P_0 \,\&\, Q_0),$$
$$U_0 = P_0 \wedge Q_0,$$
$$V_2 = ED_2 \wedge E_3 \wedge E_4 \wedge ((R_1 \,\&\, S_1) \mid ((R_1 \mid S_1) \,\&\, R_0 \,\&\, S_0)),$$
$$V_1 = R_1 \wedge S_1 \wedge (R_0 \,\&\, S_0),$$
$$V_0 = R_0 \wedge S_0,$$
$$E = (U_2 \,\&\, V_2) \mid ((U_2 \mid V_2) \,\&\, U_1 \,\&\, V_1) \mid ((U_2 \mid V_2) \,\&\, (U_1 \mid V_1) \,\&\, U_0 \,\&\, V_0),$$

$$ET_1 = (ED_1 \mid (E_1 \wedge E_2 \wedge ER_1)) \,\&\, (E_1 \mid E_2 \mid \sim ER_1),$$
$$ET_2 = (ED_2 \mid (E_3 \wedge E_4 \wedge ER_2)) \,\&\, (E_3 \mid E_4 \mid \sim ER_2). \qquad (73)$$

A. Type 4, Filter 1: $R = (A_1 + A_2 + \ldots + A_{15} + A_{16} + 8*ONE) >> 4$

i. Conventional SIMD Solution

[93] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (16 Instructions) A_{iL} = Unpack Low 4 Bytes of A_i, for $i = 1,\ldots,16$,
2. (16 Instructions) A_{iH} = Unpack High 4 Bytes of A_i, for $i = 1,\ldots,16$,
3. (17 Instructions) Add and Shift lower 4 words of $A_1,\ldots,A_{16}$ to obtain lower 4 words of R_L as:

   $R_L = (A_{1L} + A_{2L} + \ldots + A_{15L} + A_{16L} + 4*ONE_4) >> 4$,
4. (17 Instructions) Add and Shift higher 4 words of $A_1,\ldots,A_{16}$ to obtain higher 4 words of R_H as:

   $R_H = (A_{1H} + A_{2H} + \ldots + A_{15H} + A_{16H} + 4*ONE_4) >> 4$,
5. (1 Instruction) Pack R_H and R_L into final register R.

We require 67 instructions to compute this filter by conventional SIMD methods.

ii. Efficient SIMD Solution

[94] In order to implement this filter efficiently, we simplify it as follows:

$$R = ((A_1+A_2+\ldots+A_7+A_8+4*ONE)>>3 + (A_9+A_{10}+\ldots+A_{15}+A_{16}+4*ONE)>>3 + E) >> 1\,, \quad (74)$$

where E is the error/correction term that is necessary for dividing up the expression into two parts each containing a Type 3, Filter 1 (42). Note that according to (42) and the registers in (73), we have:

$$(A_1+A_2+\ldots+A_7+A_8+4*ONE)>>3 = D_1 - (ET_1 \,\&\, ONE),$$
$$(A_9+A_{10}+\ldots+A_{15}+A_{16}+4*ONE)>>3 = D_2 - (ET_2 \,\&\, ONE), \quad (75)$$

where ET_1 and ET_2 are error/correction terms obtained in (42). We can simplify (74) as:

$$R = (D_1 + D_2 + (E - ET_1 - ET_2)\&ONE) >> 1. \quad (76)$$

Note that the expressions for E, ET_1, and ET_2 are given in (73). We note that $E, ET_1, ET_2 \in \{0, 1\}$, and $E_T = (E - ET_1 - ET_2) \in \{-2, -1, 0, 1\}$. From (76), we have:

$$R = \begin{cases} PAVG(D_1,D_2) - ONE - (D_1 \wedge D_2)\ \&\ ONE & \text{when } E_T = -2 \\ PAVG(D_1,D_2) - ONE & \text{when } E_T = -1 \\ PAVG(D_1,D_2) - (D_1 \wedge D_2)\ \&\ ONE & \text{when } E_T = 0 \\ PAVG(D_1,D_2) & \text{when } E_T = 1 \end{cases}. \tag{77}$$

We simplify (77) as:

$$R = PAVG(D_1,D_2) - ((ET_1\ \&\ ET_2)\ |\ {\sim}E)\ \&\ (ET_1\ |\ ET_2\ |\ E)\ \&\ ONE$$
$$- (D_1 \wedge D_2)\ \&\ {\sim}(ET_1 \wedge ET_2 \wedge E)\ \&\ ONE. \tag{78}$$

We can further simplify (78) as:

U_1 = At least 1 *ED*,

U_2 = At least 1 *EC*,

U_3 = At least 1 *EB*,

U_4 = Both *ED*s,

U_5 = At least 2 *EC*s,

U_6 = At least 3 *EC*s,

U_7 = All 4 *EC*s,

U_8 = At least 3 *EB*s,

U_9 = At least 5 *EB*s,

U_{10} = At least 7 *EB*s,

U_{11} = At least 1 *ED, EC* or *EB,*

$$E_1 = (EX\ \&\ U_{11})\ |\ (U_4\ \&\ (U_2\ |\ U_3))\ |\ (U_1\ \&\ U_6)\ |\ (U_1\ \&\ U_5\ \&\ U_3)\ |\ (U_1\ \&\ U_2\ \&\ U_8)\ |$$
$$(U_1\ \&\ U_9)\ |\ (U_7\ \&\ U_3)\ |\ (U_6\ \&\ U_8)\ |\ (U_5\ \&\ U_9)\ |\ (U_2\ \&\ U_{10}),$$
$$E_2 = (EX\ \&\ U_4\ \&\ ((U_7\ \&\ U_3)\ |\ (U_6\ \&\ U_8)\ |\ (U_5\ \&\ U_9)\ |\ (U_2\ \&\ U_{10})))\ |$$
$$(EX\ \&\ U_1\ \&\ ((U_7\ \&\ U_9)\ |\ (U_6\ \&\ U_{10})),$$
$$E = E_1 + E_2. \tag{79}$$

[95] Clearly, (79) is an inefficient solution and is useful in special cases and for approximate solutions only.

iii. Approximate SIMD Solution

[96] We have many approximate solutions by assuming the least significant bit of EB_1, ..., EB_8, EC_1, ... EC_4, ED_1, or ED_2 as 0 or 1. With the assumption the last bit of $E = ET_1 = ET_2 = 1$, we get from (78):

$$R = PAVG(D_1,D_2) - ONE. \tag{80}$$

This solution requires 16 instructions, and produces a maximum error of ±1 in the final result for 8.6% of all possible values of $A_1,...,A_{16}$ between [0,255]. The error never exceeds ±1. We receive a computational advantage of 67:16, and approximate 4 times speedup.

B. Type 4, Special Filter 1: R = (A$_1$ + 4A$_2$ + 6A$_3$ + 4A$_4$ + A$_5$ + 8*ONE) >> 4

[97] This filter is a Gaussian approximation filter used for post-processing.

i. Conventional SIMD Solution

[98] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (5 Instructions) $A_{i\mathrm{L}}$ = Unpack Low 4 Bytes of A_i, for $i \in \{1,2,3,4,5\}$,
2. (5 Instructions) $A_{i\mathrm{H}}$ = Unpack High 4 Bytes of A_i, for $i \in \{1,2,3,4,5\}$,
3. (9 Instructions) Add and Shift lower 4 words of A_1, ..., A_5 to obtain lower 4 words of R_L as:

   $R_\mathrm{L} = (A_{1\mathrm{L}} + 4A_{2\mathrm{L}} + 6A_{3\mathrm{L}} + 4A_{4\mathrm{L}} + A_{5\mathrm{L}} + 8{*}ONE_4) >> 4$,
4. (9 Instructions) Add and Shift higher 4 words of A_1, ..., A_5 to obtain higher 4 words of R_H as:

   $R_\mathrm{H} = (A_{1\mathrm{H}} + 4A_{2\mathrm{H}} + 6A_{3\mathrm{H}} + 4A_{4\mathrm{H}} + A_{5\mathrm{H}} + 8{*}ONE_4) >> 4$,
5. (1 Instruction) Pack R_H and R_L into final register R.

We require 29 instructions to compute this filter by conventional SIMD methods.

ii. Efficient SIMD Solution

From (73), we get the:

$$B_1 = PAVG(A_1,A_2), B_2 = A_3,\ B_3 = A_3, B_4 = A_3, B_5 = A_4, B_6 = A_4,\ B_7 = A_5, B_8 = A_5,$$
$$C_1 = PAVG(B_1,A_3), C_2 = A_3, C_3 = A_4, C_4 = A_5,$$
$$D_1 = PAVG(C_1,A_3), D_2 = PAVG(A_4,A_5),$$
$$EB_1 = A_1\text{^}A_2, EB_2 = EB_3 = EB_4 = EB_5 = EB_6 = EB_7 = EB_8 = 0,$$
$$EC_1 = (B_1\text{^}A_3), EC_2 = EC_3 = EC_4 = 0,$$
$$ED_1 = C_1\text{^}A_3, ED_2 = A_4\text{^}A_5,$$

$$E_1 = EC_1\ \&\ EB_1, E_2 = E_3 = E_4 = 0,$$
$$P_1 = EC_1\ \text{^}\ {\sim}EB_1,\ P_0 = EB_1, Q_1 = 1, Q_0 = 0, ER_1 = P_1,$$
$$R_1 = 1,\ R_0 = 0, S_1 = 1, S_0 = 0, ER_2 = 1,$$
$$U_2 = ED_1\ \text{^}\ E_1\ \text{^}\ P_1, U_1 = {\sim}P_1, U_0 = P_0,$$
$$V_2 = {\sim}ED_2, V_1 = 0, V_0 = 0,$$
$$E = U_2\ \&\ V_2,$$
$$ET_1 = (ED_1\ |\ (E_1\ \text{^}\ ER_1)), ET_2 = 0. \tag{81}$$

From (78) we get:

$$R = PAVG(D_1,D_2) - ({\sim}E\ \&\ ET_1\ \&\ ONE) - (D_1\text{^}D_2)\ \&\ {\sim}(ET_1\text{^}E)\ \&\ ONE. \tag{82}$$

We can simplify (82) as follows:

$$U = EC_1\ |\ EB_1,$$
$$R = PAVG(D_1,D_2) - (((D_1\text{^}D_2)\ \&\ (ED_1\ |\ ED_2\ |\ U))\ |\ (ED_1\ \&\ ED_2\ \&\ U))\ \&\ ONE. \tag{83}$$

The solution in (83) requires 19 instructions with a 29:19 computational advantage.

iii. Approximate SIMD Solution

[99] We can assume the least significant bit of ED_1, ED_2, EC_1, or EB_1 as 0 or 1 to get several approximate solutions. We first make the assumption that the least significant bit of ED_1=1, and $ED_2 = EC_1 = EB_1 = 0$, to get the following solution:

$$R = PAVG(D_1,D_2) - (D_1\text{^}D_2)\ \&\ ONE. \tag{84}$$

This solution requires 8 instructions, and produces a maximum error of ±1 for 12.5% of all possible values of $A_1,\ldots,A_5$ between [0,255]. The computational advantage is 29:8 (more than 3 times speedup).

The second approximate solution makes the assumption that the least significant bit of $EC_1 = 1$, and $EB_1 = 0$, to get the solution:

$$R = PAVG(D_1,D_2) - ((D_1 \wedge D_2) \mid (ED_1 \ \& \ ED_2)) \ \& \ ONE. \tag{85}$$

This solution requires 12 instructions, and produces a maximum error of ±1 for 6.25% of all possible values of $A_1,\ldots,A_5$ between [0,255]. The computational advantage is 29:12.

C. Type 4, Special Filter 2:

$R = (A_1+A_2+2A_5+2A_6+2A_7+2A_8+4A_9+A_3+A_4+8*ONE) >> 4$

[100] This filter is also a Gaussian approximation filter used for post-processing.

i. Conventional SIMD Solution

[101] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (9 Instructions) $A_{i\mathrm{L}}$ = Unpack Low 4 Bytes of A_i, for $i \in \{1,2,\ldots,9\}$,
2. (9 Instructions) $A_{i\mathrm{H}}$ = Unpack High 4 Bytes of A_i, for $i \in \{1,2,\ldots,9\}$,
3. (15 Instructions) Add and Shift lower 4 words of $A_1, \ldots, A_5$ to obtain lower 4 words of R_L as:

   $R_\mathrm{L} = (A_{1\mathrm{L}} + A_{2\mathrm{L}} + 2A_{5\mathrm{L}} + 2A_{6\mathrm{L}} + 2A_{7\mathrm{L}} + 2A_{8\mathrm{L}} + 4A_{9\mathrm{L}} + A_{3\mathrm{L}} + A_{4\mathrm{L}} + 8*ONE_4) >> 4$,
4. (15 Instructions) Add and Shift higher 4 words of $A_1, \ldots, A_5$ to obtain higher 4 words of R_H as:

   $R_\mathrm{H} = (A_{1\mathrm{H}} + A_{2\mathrm{H}} + 2A_{5\mathrm{H}} + 2A_{6\mathrm{H}} + 2A_{7\mathrm{H}} + 2A_{8\mathrm{H}} + 4A_{9\mathrm{H}} + A_{3\mathrm{H}} + A_{4\mathrm{H}} + 8*ONE_4) >> 4$,
5. (1 Instruction) Pack R_H and R_L into final register R.

We require 49 instructions to compute this filter by conventional SIMD methods.

ii. Efficient SIMD Soluti n

[102] From (73), we get the:

$$B_1 = PAVG(A_1,A_2),\ B_2 = PAVG(A_3,A_4),\ \ B_3 = A_5,\ B_4 = A_6,\ B_5 = A_7,\ B_6 = A_8,\ \ B_7 = A_9,\ B_8 = A_9,$$

$$C_1 = PAVG(B_1,B_2),\ C_2 = PAVG(A_5,A_6),\ C_3 = PAVG(A_7,A_8),\ C_4 = A_9,$$

$$D_1 = PAVG(C_1,C_2),\ D_2 = PAVG(C_3,A_9),$$

$$EB_1 = (A_1 \hat{}\ A_2),\ EB_2 = (A_3 \hat{}\ A_4),\ EB_3 = EB_4 = EB_5 = EB_6 = EB_7 = EB_8 = 0,$$

$$EC_1 = (B_1 \hat{}\ B_2),\ EC_2 = (A_5 \hat{}\ A_6),\ EC_3 = (A_7 \hat{}\ A_8),\ EC_4 = 0,$$

$$ED_1 = (C_1 \hat{}\ C_2),\ ED_2 = (C_3 \hat{}\ A_9),$$

$$E_1 = EC_1\ \&\ (EB_1 \mid EB_2),\ E_2 = E_3 = E_4 = 0,$$

$$P_1 = EC_1\ \hat{}\ \sim(EB_1 \mid EB_2),\ \ P_0 = (EB_1 \hat{}\ EB_2),\ Q_1 = \sim EC_2,\ Q_0 = 0,\ ER_1 = P_1\ \&\ Q_1,$$

$$R_1 = \sim EC_3,\ \ R_0 = 0,\ S_1 = 1,\ S_0 = 0,\ ER_2 = R_1,$$

$$U_2 = ED_1\ \hat{}\ E_1\ \hat{}\ (P_1\ \&\ Q_1),\ U_1 = P_1\ \hat{}\ Q_1,\ U_0 = P_0,$$

$$V_2 = ED_2\ \hat{}\ R_1,\ V_1 = \sim R_1,\ V_0 = 0,$$

$$E = (U_2\ \&\ V_2) \mid ((U_2 \mid V_2)\ \&\ U_1\ \&\ V_1),$$

$$ET_1 = (ED_1 \mid (E_1\ \hat{}\ ER_1))\ \&\ (E_1 \mid \sim ER_1),\ ET_2 = (ED_2\ \&\ \sim ER_2). \tag{86}$$

[103] The final solution is same as (78). We can simplify this solution as follows:

$$U = EB_1 \mid EB_2,$$

$$V = EC_1 \mid EC_3,$$

$$W = EC_2 \mid U \mid V,$$

$$Z = ED_1 \mid ED_2,$$

$$F = Z \mid W,$$

$$H = EC_1\ \&\ EC_3,$$

$$G = (EC_2\ \&\ V) \mid H,$$

$$R = PAVG(D_1,D_2) - (((D_1 \hat{}\ D_2)\&F) \mid (ED_1\&ED_2\&W) \mid (Z\ \&\ ((EC_2\&H)|(G\&U))))\ \&\ ONE. \tag{87}$$

[104] The solution in (87) requires 36 instructions with a 49:36 computational advantage.

iii. Approximate SIMD Soluti n

[105] We suggest 2 approximate solutions for this filter. For the first approximate solution, we assume the least significant bit of EC_3=0, and EB_1= EB_2=1 to get the following:

$$R = PAVG(D_1,D_2) - ((D_1 \wedge D_2) \mid (ED_1 \& ED_2) \mid ((ED_1 | ED_2) \ \& \ EC_1 \ \& \ EC_2)) \ \& \ ONE. \quad (88)$$

This solution requires 21 instructions, and produces a maximum error of ±1 for 6.25% of all possible values of $A_1,...,A_9$ between [0,255]. The computational advantage is 49:21 (more than 2 times speedup).

[106] The second approximate solution makes the assumption that the least significant bit of $EB_1 = EB_2 = 1$. We get the solution:

$$U = (EC_1 \ \& \ (EC_2 \mid EC_3)) \mid (EC_2 \ \& \ EC_3),$$

$$R = PAVG(D_1,D_2) - ((D_1 \wedge D_2) \mid (ED_1 \& ED_2) \mid ((ED_1 | ED_2) \ \& \ U)) \ \& \ ONE. \quad (89)$$

This solution requires 25 instructions, and produces a maximum error of ±1 for 3.12% of all possible values of $A_1,...,A_9$ between [0,255]. The computational advantage is 49:25 (nearly 2 times speedup).

D. Type 4, Special Filter 3:

$R = (A_1+2A_2+2A_3+2A_4+2A_5+2A_6+2A_7+2A_8+A_9+8*ONE) >> 4$

[107] This filter is also used for post-processing.

i. Conventional SIMD Solution

[108] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (9 Instructions) A_{iL} = Unpack Low 4 Bytes of A_i, for $i \in \{1,2,...,9\}$,
2. (9 Instructions) A_{iH} = Unpack High 4 Bytes of A_i, for $i \in \{1,2,...,9\}$,
3. (17 Instructions) Add and Shift lower 4 words of A_1, ..., A_5 to obtain lower 4 words of R_L as:

   $R_L = (A_{1L} + 2A_{2L} + 2A_{3L} + 2A_{4L} + 2A_{5L} + 2A_{6L} + 2A_{7L} + 2A_{8L} + A_{9L} + 8*ONE_4) >>$
4,

4. (17 Instructions) Add and Shift higher 4 words of A_1, ..., A_5 to obtain higher 4 words of R_H as:

$R_H = (A_{1H} + 2A_{2H} + 2A_{3H} + 2A_{4H} + 2A_{5H} + 2A_{6H} + 2A_{7H} + 2A_{8H} + A_{9H} + 8*ONE_4) >>$
4,

5. (1 Instruction) Pack R_H and R_L into final register R.

We require 53 instructions to compute this filter by conventional SIMD methods.

ii. Efficient SIMD Solution

[109] From (73), we get the:

$$B_1 = PAVG(A_1,A_2),\ B_2 = A_2,\ B_3 = A_3,\ B_4 = A_4,\ B_5 = A_5,\ B_6 = A_6,\ B_7 = A_7,\ B_8 = A_8,$$
$$C_1 = PAVG(B_1,A_2),\ C_2 = PAVG(A_3,A_4),\ C_3 = PAVG(A_5,A_6),\ C_4 = PAVG(A_7,A_8),$$
$$D_1 = PAVG(C_1,C_2),\ D_2 = PAVG(C_3,C_4),$$
$$EB_1 = (A_1 \wedge A_9),\ EB_2 = EB_3 = EB_4 = EB_5 = EB_6 = EB_7 = EB_8 = 0,$$
$$EC_1 = (B_1 \wedge A_2),\ EC_2 = (A_3 \wedge A_4),\ EC_3 = (A_5 \wedge A_6),\ EC_4 = (A_7 \wedge A_8),$$
$$ED_1 = (C_1 \wedge C_2),\ ED_2 = (C_3 \wedge C_4),$$

$$E_1 = EC_1 \ \&\ EB_1,\ E_2 = E_3 = E_4 = 0,$$
$$P_1 = EC_1 \wedge {\sim}EB_1,\ P_0 = EB_1,\ Q_1 = {\sim}EC_2,\ Q_0 = 0,\ ER_1 = P_1 \ \&\ Q_1,$$
$$R_1 = {\sim}EC_3,\ R_0 = 0,\ S_1 = {\sim}EC_4,\ S_0 = 0,\ ER_2 = R_1 \ \&\ S_1,$$

$$U_2 = ED_1 \wedge E_1 \wedge (P_1 \ \&\ Q_1),\ U_1 = P_1 \wedge Q_1,\ U_0 = P_0,$$
$$V_2 = ED_2 \wedge (R_1 \ \&\ S_1),\ V_1 = R_1 \wedge S_1,\ V_0 = 0,$$
$$E = (U_2 \ \&\ V_2) \mid ((U_2 \mid V_2) \ \&\ U_1 \ \&\ V_1),$$
$$ET_1 = (ED_1 \mid (E_1 \wedge ER_1)) \ \&\ (E_1 \mid {\sim}ER_1),\ ET_2 = (ED_2 \ \&\ {\sim}ER_2). \qquad (90)$$

The final solution is same as (78). We can simplify this solution as follows:

$$U_1 = EC_1 \ \&\ EC_2,$$
$$U_2 = EC_3 \ \&\ EC_4,$$
$$U_3 = ED_1 \ \&\ ED_2,$$
$$V_1 = EC_1 \mid EC_2,$$

$$V_2 = EC_3 \mid EC_4,$$
$$V_3 = ED_1 \mid ED_2,$$
$$V_4 = V_2 \mid EB_1,$$
$$W = (V_1 \,\&\, V_2 \,\&\, EB_1) \mid (U_1 \,\&\, V_4) \mid (U_2 \,\&\, (V_1 \mid EB_1)),$$
$$F = V_1 \mid V_4,$$
$$E = D_1 \wedge D_2,$$
$$G = U_1 \,\&\, U_2 \,\&\, EB_1,$$
$$H = G \,\&\, U_3 \,\&\, E,$$
$$R = PAVG(D_1,D_2) - (((E \,\&\, (V_3 \mid F)) \mid (U_3 \,\&\, F) \mid (V_3 \,\&\, W) \mid G) \,\&\, ONE) - (H \,\&\, ONE). \tag{91}$$

[110] The solution in (91) requires 46 instructions with a 53:46 computational advantage.

iii. Approximate SIMD Solution

[111] We suggest 2 approximate solutions for this filter. For the first approximate solution, we assume the least significant bit of $EC_1 = EC_2 = 1$, and $EC_3 = EC_4 = 0$ to get the following:

$$R = PAVG(D_1,D_2) - ((D_1 \wedge D_2) \mid (ED_1 \,\&\, ED_2) \mid ((ED_1 \mid ED_2) \,\&\, EB_1)) \,\&\, ONE. \tag{92}$$

This solution requires 19 instructions, and produces a maximum error of ±1 for 9.38% of all possible values of $A_1,\ldots,A_9$ between [0,255]. The computational advantage is 53:19 (more than 3 times speedup).

The second approximate solution makes the assumption that the least significant bit of $EC_1 = 1$, and $EC_3 = 0$. We get the solution:

$$W = (EC_4 \,\&\, EB_1) \mid (EC_2 \,\&\, (EC_4 \mid EB_1)),$$
$$R = PAVG(D_1,D_2) - ((D_1 \wedge D_2) \mid (ED_1 \,\&\, ED_2) \mid ((ED_1 \mid ED_2) \,\&\, W)) \,\&\, ONE. \tag{93}$$

[112] This solution requires 25 instructions, and produces a maximum error of ±1 for 6.25% of all possible values of $A_1,\ldots, A_9$ between [0,255]. The computational advantage is 53:25 (more than 2 times speedup).

E. Type 4, Special Filter 4: R = (A$_1$+2A$_2$+3A$_3$+4A$_4$+3A$_5$+2A$_6$+A$_7$+8*ONE) >> 4

[113] This filter is also used for post-processing.

i. Conventional SIMD Solution

[114] This filter can be implemented in SIMD architecture (assuming sufficient memory) by using the following steps:

1. (7 Instructions) $A_{i\mathrm{L}}$ = Unpack Low 4 Bytes of A_i, for $i \in \{1,2,\ldots,7\}$,
2. (7 Instructions) $A_{i\mathrm{H}}$ = Unpack High 4 Bytes of A_i, for $i \in \{1,2,\ldots,7\}$,
3. (13 Instructions) Add and Shift lower 4 words of A_1, ..., A_5 to obtain lower 4 words of R_L as:

   $R_\mathrm{L} = (A_{1\mathrm{L}} + 2A_{2\mathrm{L}} + 3A_{3\mathrm{L}} + 4A_{4\mathrm{L}} + 3A_{5\mathrm{L}} + 2A_{6\mathrm{L}} + A_{7\mathrm{L}} + 8{*}ONE_4) >> 4$,
4. (13 Instructions) Add and Shift higher 4 words of A_1, ..., A_5 to obtain higher 4 words of R_H as:

   $R_\mathrm{H} = (A_{1\mathrm{H}} + 2A_{2\mathrm{H}} + 3A_{3\mathrm{H}} + 4A_{4\mathrm{H}} + 3A_{5\mathrm{H}} + 2A_{6\mathrm{H}} + A_{7\mathrm{H}} + 8{*}ONE_4) >> 4$,
5. (1 Instruction) Pack R_H and R_L into final register R.

We require 41 instructions to compute this filter by conventional SIMD methods.

ii. Efficient SIMD Solution

[115] From (73), we get the:

$$B_1 = PAVG(A_1,A_7), B_2 = A_2,\ B_3 = A_4, B_4 = A_4, B_5 = A_3, B_6 = A_5,\ B_7 = A_6, B_8 = PAVG(A_3,A_5),$$

$$C_1 = PAVG(B_1,A_2), C_2 = A_4, C_3 = PAVG(A_3,A_5), C_4 = PAVG(A_6,B_8),$$

$$D_1 = PAVG(C_1,A_4), D_2 = PAVG(C_3,C_4),$$

$$EB_1 = A_1 \wedge A_7, EB_2 = EB_3 = EB_4 = EB_5 = EB_6 = EB_7 = 0, EB_8 = A_3 \wedge A_5,$$

$$EC_1 = B_1 \wedge A_2, EC_2 = 0, EC_3 = A_3 \wedge A_5, EC_4 = A_6 \wedge B_8,$$

$$ED_1 = C_1 \wedge A_4, ED_2 = C_3 \wedge C_4,$$

$$E_1 = EC_1\ \&\ EB_1, E_2 = E_3 = 0, E_4 = EC_4\ \&\ EB_8,$$

$$P_1 = EC_1 \wedge {\sim}EB_1,\ P_0 = EB_1, Q_1 = 1, Q_0 = 0, ER_1 = P_1,$$

$$R_1 = \sim EC_3,\ R_0 = 0, S_1 = EC_4 \wedge \sim EB_8, S_0 = EB_8, ER_2 = R_1 \,\&\, S_1,$$

$$U_2 = ED_1 \wedge E_1 \wedge P_1, U_1 = \sim P_1, U_0 = P_0,$$
$$V_2 = ED_2 \wedge E_4 \wedge (R_1 \,\&\, S_1), V_1 = R_1 \wedge S_1, V_0 = S_0,$$
$$E = (U_2 \,\&\, V_2) \mid ((U_2 \mid V_2) \,\&\, U_1 \,\&\, V_1) \mid ((U_2 \mid V_2) \,\&\, (U_1 \mid V_1) \,\&\, U_0 \,\&\, V_0),$$
$$ET_1 = (ED_1 \mid (E_1 \wedge P_1)) \,\&\, (E_1 \mid \sim P_1), ET_2 = (ED_2 \mid (E_4 \wedge ER_2)) \,\&\, (E_4 \mid \sim ER_2). \tag{94}$$

The final solution is same as (78). We can simplify this solution as follows:

$$U_1 = EC_3 \mid EC_4,$$
$$U_2 = EB_1 \mid EB_8,$$
$$U_3 = ED_1 \mid ED_2,$$
$$U_4 = EC_1 \mid U_1,$$
$$U_5 = U_4 \mid U_2,$$
$$U_6 = U_5 \mid U_3,$$
$$U_7 = EC_3 \,\&\, EC_4,$$
$$U_8 = (EC_1 \,\&\, U_1) \mid U_7,$$
$$U_9 = (EC_1 \,\&\, U_7) \mid (U_8 \,\&\, U_2),$$
$$R = PAVG(D_1,D_2) - (((D_1 \wedge D_2) \,\&\, U_6) \mid (ED_1 \,\&\, ED_2 \,\&\, U_5) \mid (U_3 \,\&\, U_9)) \,\&\, ONE. \tag{95}$$

The solution in (95) requires 36 instructions with a 41:36 computational advantage.

iii. Approximate SIMD Solution

[116] We suggest 2 approximate solutions for this filter. For the first approximate solution, we assume the least significant bit of $EC_1 = EC_3 = 1$, and $EC_4 = EB_1 = 0$ to get the following:

$$R = PAVG(D_1,D_2) - ((D_1 \wedge D_2) \mid (ED_1 \,\&\, ED_2) \mid ((ED_1 \mid ED_2) \,\&\, EB_8)) \,\&\, ONE. \tag{96}$$

This solution requires 19 instructions, and produces a maximum error of ±1 for 6.25% of all possible values of $A_1,\ldots,A_7$ between [0,255]. The computational advantage is 41:19 (more than 2 times speedup).

[117] The second approximate solution makes the assumption that the least significant bit of $EC_3 = 1$, and $EB_1 = 0$. We get the solution:

$$U_9 = (EC_1 \ \& \ EC_4) \mid ((EC_1 \mid EC_4) \ \& \ EB_8),$$

$$R = PAVG(D_1,D_2) - ((D_1 \wedge D_2) \mid (ED_1 \ \& \ ED_2) \mid ((ED_1 \mid ED_2) \ \& \ U_9)) \ \& \ ONE. \quad (97)$$

This solution requires 25 instructions, and produces a maximum error of ±1 for 3.13% of all possible values of $A_1,\ldots,A_7$ between [0,255]. The computational advantage is 41:25.

F. Type 4, Summary of Results

[118] Table VI below summarizes the instructions required to compute each filter (given sufficient memory) by the efficient and conventional SIMD methods. For the Efficient case, we give the instructions required for the exact and approximate solutions.

Summary of Results for Type 4 FIR Filters.

Type 4 Filters	Conventional Method	Efficient Method		Speedup	
		Exact	Approx.	Exact	Approx.
$(A_1 + A_2 + \ldots + A_{16} + 8*ONE) >> 4$	67	N/A	16	N/A	4.2
$(A_1+4A_2+6A_3+4A_4+A_5+8*ONE)>>4$	29	19	8	1.5	3.6
$(A_1+A_2+2A_5+2A_6+2A_7+2A_8+4A_9+A_3+A_4+8*ONE)>>4$	49	36	21	1.4	2.3
$(A_1+2A_2+2A_3+2A_4+2A_5+2A_6+2A_7+2A_8+A_9+8*ONE)>>4$	53	46	19	1.2	2.8
$(A_1+2A_2+3A_3+4A_4+3A_5+2A_6+A_7+8*ONE)>>4$	41	36	19	1.1	2.2

TABLE VI

[119] The shaded areas show significant improvements in efficiency due to the analyses developed here.

5. Corrected Approximation Method

[120] Another approach, referred to as the "corrected approximation" method, can be used to improve upon the approaches described in sections 3 and 4 when exact computation is required. Steps of the corrected approximation method include the following:

1. Use *PAVG* to quickly get an approximation to the correct result.
2. Adjust the approximation so that it becomes no less than the correct result.

3. Perform another computation to determine the value of the least significant bits of the correct result.
4. Use the value of the least significant bits from step 3 to determine the error of the value obtained from step 2.
5. Subtract the error value in step 4 from the approximate value in step 2 in order to get the correct final result.

[121] Examples of the corrected approximation method are given in the following subsections for type 3 and type 4 filters, respectively.

A. Corrected Approximation Method Applied to Type 3 Filters

[122] As in section 3, we define the following packed 64-bit registers, each containing 8 data elements of one byte each:

$$B_1 = PAVG(A_1,A_2), B_2 = PAVG(A_3,A_4), \; B_3 = PAVG(A_5,A_6), B_4 = PAVG(A_7,A_8),$$
$$C_1 = PAVG(B_1,B_2), C_2 = PAVG(B_3,B_4), D = PAVG(C_1,C_2). \qquad (98)$$

D is computed in 7 operations, and is an approximate solution to all filters of the form:

$R = (A_1 + A_2 + A_3 + A_4 \; + A_5 + A_6 + A_7 + A_8 + c{*}ONE) >> 3, \textit{where } 0 \leq c \leq$
4

[123] It can be determined how far off D can be from the correct value of R. The proof is omitted for the purpose of brevity, but the answer is $R-1 \leq D \leq R+2$.

[124] After D is computed, the second step is to increase D by 1, so that D is at least as big as R. However, one must be cautious on this step: if a byte holds the value of 255 and it is increased by 1, then most architectures automatically clip the sum of 256 so that it becomes 0, which is not what we want. This is remedied by using a "saturated add", which will add 1 to each packed byte only if that byte is less than 255. All bytes that are 255 remain the same. The instruction for this is PADDUSB:

$$PADDUSB(D, 1) \qquad (99)$$

Thus, after 8 instructions, we have computed a value D satisfying $R \leq D \leq R+3$.

[125] The third step is to determine the correct least significant bits of R. This is done by performing the computation of R as it is defined:

$$L = CLIP(A_1 + A_2 + A_3 + A_4 \; + A_5 + A_6 + A_7 + A_8 + c{*}ONE) >> 3. \qquad (100)$$

The value $c*ONE$ is a stored constant similar to ONE, so there is no need to perform a multiply. Also, most architecture perform the $CLIP$ automatically, so this does not count as an extra instruction. In total there are 8 adds and 1 shift to compute L, which holds the 5 least significant bits of R for each packed byte.

[126] The fourth step uses L to determine the error term for D. Since we know that D is at most 3 more than R, we only need to figure out how much needs to be subtracted from D so that it agrees with L in the two least significant bits. This is accomplished by:

$$E = CLIP(D - L) \ \& \ THREE \qquad (101)$$

As before, the $CLIP$ comes for free on most architectures and $THREE$ is a stored constant similar to ONE. In 2 instructions, the error term has been determined. The final step is to subtract this error from D to get the final result. This is 1 additional instruction.

$$R = D - E \qquad (102)$$

In total this is 20 instructions, or 19 when $c = 0$ since an addition can be saved. Pseudo-code for the approach using 20 instructions is shown in Table VII, below.

```
#define P(a,b)    (((a) + (b) + 1) >> 1)
b1 = P(a01,a02);
b2 = P(a03,a04);
b3 = P(a05,a06);
b4 = P(a07,a08);
c1 = P(b1 ,b2 );
c2 = P(b3 ,b4 );
d  = P(c1 ,c2 );
d  = ((int)d + 1) < 256 ? d+1 : 255; /* saturated add */
e  = (a01 + a02 + a03 + a04 + a05 + a06 + a07 + a08 + cc) >> 3;
dd = (d - e) & 0x03;
x1 = d - dd;
```

TABLE VII

[127] The approach of Table VII provides an exact answer in 20 instructions compared to 32 of the approach of Table III. In fact, the exact solution provided by Table VII even beats the approximate solution of Table VI (22 instructions). However, the approximate solution offers significant advantages in many special filter computations.

[128] Further refinements can be obtained when certain operands are repeated. For instance, consider the following filter:

$$(A_1+A_2+2A_3+2A_5+2A_7+4*ONE)>>3$$

[129] There is no need to do $PAVG(A_3,A_3)$, $PAVG(A_5,A_5)$, or $PAVG(A_7,A_7)$ since any $PAVG$ of a number with itself is just itself. This saves 3 instructions. Further, the computation of L can be shortened by first doing $(A_3+A_5+A_7) << 1$ and then adding on the remaining parts. Hence there are only 5 additions, but 1 additional shift instruction. The net savings on L is 2 instructions, and the total net savings is 5 instructions.

[130] Our results are summarized in Table VIII below.

Summary of Results for Type 3 FIR Filters.

Type 3 Filters	**Section 3 Method**	**Corrected Approximation Method**
$(A_1+A_2+\ldots+A_8+4*ONE)>>3$	32	20
$(A_1+A_2+\ldots+A_8+3*ONE)>>3$	35	20
$(A_1+A_2+\ldots+A_8+2*ONE)>>3$	34	20
$(A_1+A_2+\ldots+A_8+1*ONE)>>3$	35	20
$(A_1+A_2+\ldots+A_8)>>3$	34	19
$(A_1+2A_3+2A_5+2A_7+A_2+4*ONE)>>3$	16	15
$(A_1+A_2+A_3+3A_4+2A_7+4*ONE)>>3$	19	17
$(A_1+A_2+A_3+2A_4+A_5+A_6+A_7+4*ONE)>>3$	27	19

TABLE VIII

B. Corrected Approximation Method Applied to Type 4 Filters

[131] Following section 4, we define the following packed 64-bit registers, each containing 8 data elements of one byte each:

$$B_1=PAVG(A_1,A_2),\ B_2=PAVG(A_3,A_4),\ B_3=PAVG(A_5,A_6),\ B_4=PAVG(A_7,A_8),$$
$$B_5=PAVG(A_9,A_{10}),\ B_6=PAVG(A_{11},A_{12}),\ B_7=PAVG(A_{13},A_{14}),\ B_8=PAVG(A_{15},A_{16}),$$
$$C_1=PAVG(B_1,B_2),\ C_2=PAVG(B_3,B_4),\ C_3=PAVG(B_5,B_6),\ C_4=PAVG(B_7,B_8),$$

$$D_1 = PAVG(C_1,C_2), D_2 = PAVG(C_3,C_4), D = PAVG(D_1,D_2) \quad (103)$$

D is computed in 15 operations, and is an approximate solution to all filters of the form:

$R = (A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8 + A_9 + A_{10} + A_{11} + A_{12} + A_{13} + A_{14} + A_{15} + A_{16} + c*ONE) >> 3$

where $0 \leq c \leq 8$. It can be shown that $R-1 \leq D \leq R+2$.

[132] The remaining steps are more or less the same as done for the type 3 filters:

$$PADDUSB(D, 1)$$

$$L = CLIP(A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8 + A_9 + A_{10} + A_{11} + A_{12} + A_{13} + A_{14} + A_{15} + A_{16} + c*ONE) >> 3$$

$$E = CLIP(D - L) \ \& \ THREE$$

$$R = D - E \quad (104)$$

[133] In total this is 36 instructions, or 35 when c = 0 since an addition can be saved. The results are summarized in Table IX below.

Summary of Results for Type 4 FIR Filters.

Type 4 Filters	**Section 5 Method**	**Corrected Approximation Method**
$(A_1 + A_2 + \ldots + A_{16} + 8*ONE) >> 4$	N/A	36
$(A_1+4A_2+6A_3+4A_4+A_5+8*ONE)>>4$	19	18
$(A_1+A_2+2A_5+2A_6+2A_7+2A_8+4A_9+A_3+A_4+8*ONE)>> 4$	36	24
$(A_1+2A_2+2A_3+2A_4+2A_5+2A_6+2A_7+2A_8+A_9+8*ONE)>> 4$	46	22
$(A_1+2A_2+3A_3+4A_4+3A_5+2A_6+A_7+8*ONE)>> 4$	36	24

TABLE IX

[134] Although the invention has been discussed with respect to specific embodiments thereof, these embodiments are merely illustrative, and not restrictive, of the invention.

For example, although a two-operand SIMD instruction has been primarily discussed, techniques and features of the invention may be applicable to other applications where the number of operands, or arguments, of a SIMD instruction are not the same as the number of variables, or values, in a formula, computation or function to be implemented with the SIMD instruction (i.e., a "mismatched" instruction).

[135] Although the invention has been described with respect to specific SIMD instructions to obtain an average of values, any other type of SIMD instruction or operation may benefit from the approach of the invention. Although specific operations such as addition, subtraction, bitwise AND, bitwise OR, bitwise logical right shift, bitwise logical left shift, bitwise exclusive OR, etc., are used in specific embodiments to achieve a result, other embodiments may use different operations, or combinations of operations, to achieve results. For example, an AND function can be realized by using an OR function and complementing, or inverting, the operands and result. Other such operational equivalents will be apparent.

[136] Alternative methods of detecting when the sum of two packed values results in an odd number can be employed. Some processors may provide instructions that combine multiple operations into compound one or more instructions. Although specific reference has been made to a "SIMD" type of instruction, other types of parallel instructions may be within the scope of the invention. Although the SIMD instruction has been described as a single instruction, other embodiments may use SIMD instructions that occupy more than a single instruction's worth of clock cycles, instruction cycles, or the like.

[137] There are various ways that the invention can be modified from specific embodiments described herein to achieve similar results. For example, adjustments to an approximate solution are performed as an intermediate step before computing the final result so that the approximate solution is no less than the actual solution. One modification can be to adjust the approximate solution so that it becomes no larger than the actual solution as an intermediate step. Such modifications will be apparent to one of skill in the art and are within the scope of the invention.

[138] Any suitable programming language can be used to implement the routines of the present invention including C, C++, Java, assembly language, etc. Different programming techniques can be employed such as procedural or object oriented. The

routines can execute on a single processing device or multiple processors. Although the steps, operations or computations may be presented in a specific order, this order may be changed in different embodiments. In some embodiments, multiple steps shown as sequential in this specification can be performed at the same time. The sequence of operations described herein can be interrupted, suspended, or otherwise controlled by another process, such as an operating system, kernel, etc. The routines can operate in an operating system environment or as stand-alone routines occupying all, or a substantial part, of the system processing.

[139] Steps can be performed in hardware or software, as desired. Note that steps can be added to, taken from or modified from the steps presented in this specification without deviating from the scope of the invention. In general, the flowcharts are only used to indicate one possible sequence of basic operations to achieve a functional aspect of the present invention.

[140] In the description herein, numerous specific details are provided, such as examples of components and/or methods, to provide a thorough understanding of embodiments of the present invention. One skilled in the relevant art will recognize, however, that an embodiment of the invention can be practiced without one or more of the specific details, or with other apparatus, systems, assemblies, methods, components, materials, parts, and/or the like. In other instances, well-known structures, materials, or operations are not specifically shown or described in detail to avoid obscuring aspects of embodiments of the present invention.

[141] A "computer-readable medium" for purposes of embodiments of the present invention may be any medium that can contain, store, communicate, propagate, or transport the program for use by or in connection with the instruction execution system, apparatus, system or device. The computer readable medium can be, by way of example only but not by limitation, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, system, device, propagation medium, or computer memory.

[142] A "processor" includes any system, mechanism or component that processes data, signals or other information. A processor can include a system with a general-purpose central processing unit, multiple processing units, dedicated circuitry for achieving functionality, or other systems. Processing need not be limited to a geographic location, or have temporal limitations. For example, a processor can perform its functions in "real time," "offline," in a "batch mode," etc. Portions of processing can be performed at different times and at different locations, by different (or the same) processing systems.

[143] Reference throughout this specification to "one embodiment", "an embodiment", or "a specific embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention and not necessarily in all embodiments. Thus, respective appearances of the phrases "in one embodiment", "in an embodiment", or "in a specific embodiment" in various places throughout this specification are not necessarily referring to the same embodiment. Furthermore, the particular features, structures, or characteristics of any specific embodiment of the present invention may be combined in any suitable manner with one or more other embodiments. It is to be understood that other variations and modifications of the embodiments of the present invention described and illustrated herein are possible in light of the teachings herein and are to be considered as part of the spirit and scope of the present invention.

[144] Embodiments of the invention may be implemented by using a programmed general purpose digital computer, by using application specific integrated circuits, programmable logic devices, field programmable gate arrays, optical, chemical, biological, quantum or nanoengineered systems, components and mechanisms may be used. In general, the functions of the present invention can be achieved by any means as is known in the art. Distributed, or networked, systems components and circuits can be used. Communication, or transfer, of data may be wired, wireless, or by any other means.

[145] It will also be appreciated that one or more of the elements depicted in the drawings/figures can also be implemented in a more separated or integrated manner, or even removed or rendered as inoperable in certain cases, as is useful in accordance with a particular application. It is also within the spirit and scope of the present invention to implement a program or code that can be stored in a machine-readable medium to permit a computer to perform any of the methods described above.

[146] Additionally, any signal arrows in the drawings/Figures should be considered only as exemplary, and not limiting, unless otherwise specifically noted. Furthermore, the term "or" as used herein is generally intended to mean "and/or" unless otherwise indicated. Combinations of components or steps will also be considered as being noted, where terminology is foreseen as rendering the ability to separate or combine is unclear.

[147] As used in the description herein and throughout the claims that follow, "a", "an", and "the" includes plural references unless the context clearly dictates otherwise. Also, as used in the description herein and throughout the claims that follow, the meaning of "in" includes "in" and "on" unless the context clearly dictates otherwise.

[148] The foregoing description of illustrated embodiments of the present invention, including what is described in the Abstract, is not intended to be exhaustive or to limit the invention to the precise forms disclosed herein. While specific embodiments of, and examples for, the invention are described herein for illustrative purposes only, various equivalent modifications are possible within the spirit and scope of the present invention, as those skilled in the relevant art will recognize and appreciate. As indicated, these modifications may be made to the present invention in light of the foregoing description of illustrated embodiments of the present invention and are to be included within the spirit and scope of the present invention.

[149] Thus, while the present invention has been described herein with reference to particular embodiments thereof, a latitude of modification, various changes and substitutions are intended in the foregoing disclosures, and it will be appreciated that in some instances some features of embodiments of the invention will be employed without a corresponding use of other features without departing from the scope and spirit of the invention as set forth. Therefore, many modifications may be made to adapt a particular situation or material to the essential scope and spirit of the present invention. It is intended that the invention not be limited to the particular terms used in following claims and/or to the particular embodiment disclosed as the best mode contemplated for carrying out this invention, but that the invention will include any and all embodiments and equivalents falling within the scope of the appended claims.